AD-A081 856 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 12/1
SPARSE GAUSSIAN ELIMINATION OF STAIRCASE LINEAR SYSTEMS.(U)
NOV 79 R FOURER N00014-75-C-0267
UNCLASSIFIED SOL-79-17 NL

1 OF 2
AD
A081856

Systems
Optimization
Laboratory

12

DTIC
ELECTE
MAR 1 1 1980
S
SCC

LEVEL II

ADA081856

Department of Operations Research
Stanford University
Stanford, CA 94305

80 3 05 042

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA
94305

6

SPARSE GAUSSIAN ELIMINATION OF STAIRCASE LINEAR SYSTEMS

by

10 Robert Fourer 14

9 TECHNICAL REPORT SOL-79-17

11 November 1979

12 122

15 N00014-75-C-0267, DE-AS03-76-SF00326

Research and reproduction of this report were partially supported by the Department of Energy Contract DE-AS03-76-SF00326 PA# DE-AT-03-76ER72018; the Office of Naval Research Contract N00014-75-C-0267; the National Science Foundation Grants MCS76-20019 A01 and ENG77-06761.

407 765

INTRODUCTION

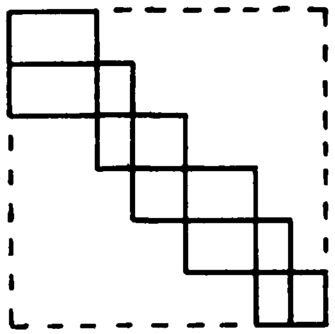
A square system of linear equations, $Bx = b$, is said to be sparse if it can be solved most efficiently through a knowledge of the arrangement of zero and nonzero elements in the matrix B. A system is proportionally sparse if the percentage of nonzeroes is significantly small; it is structurally sparse if the nonzero elements are confined to certain parts of B in a regular way. When $Bx = b$ is sparse, B is also called a sparse matrix.

The method of Gaussian elimination computes, essentially, a factorization

$$B = LU$$

where L is lower triangular and U is upper triangular. The system $Bx = b$ is thus reduced to a pair of triangular systems, $L(Ux) = b$, which are easily solved by back-substitution. Sparse Gaussian elimination denotes any adaptation of this basic method to take advantage of sparsity in B. In general, techniques of sparse elimination rely on finding an L and U that inherit B's proportional or structural sparsity.

A linear system is said to have a staircase structure if, roughly, the nonzero elements of B are confined to certain blocks on or just below the diagonal:



Staircase systems arise in models that have a natural sequence of stages --for example, planning or control models that consider similar activities at successive times. Clearly a staircase system (and the staircase matrix B) may be both proportionally and structurally sparse.

This paper proposes methods of sparse Gaussian elimination for staircase-structured systems. These methods are particularly applicable to the linear programming problem

$$\begin{array}{rl} \text{minimize} & cx \\ \text{subject to} & Ax = b, \quad \ell \leq x \leq u \end{array}$$

where the $m \times n$ matrix A has a staircase structure. The simplex method for linear programming, applied to such a problem, must solve numerous $m \times m$ staircase systems. These systems are usually quite large-- $m > 500$ is common--and are sometimes especially difficult [4,22,24]. The methods of this paper may also be applied to linear systems that arise in non-linear optimization or control, especially where the staircase blocks are dense and irregular.

The first three sections of this paper assess the current state of affairs. Section 1 reviews sparse Gaussian elimination generally, and Section 2 derives various properties of nonsingular staircase matrices. Section 3 describes the behavior of standard sparse-elimination methods on staircase systems.

The remaining sections consider special methods for staircase systems. Natural pivot orders for Gaussian elimination of staircase systems are developed in Section 4. Based on these orders, Sections 5 and 6 set forth two new approaches to staircase elimination, and report initial computational experience.

1. SPARSE GAUSSIAN ELIMINATION

Principles of elimination

The solution of $Bx = b$ does not depend on the ordering of its variables or equations. More precisely, for any choice of permutation matrices P and Q, the system $Bx = b$ is equivalent to

$$(PBQ^T)(Qx) = Pb$$

This latter system may be solved by Gaussian elimination on PBQ^T, which is just B with its rows and columns rearranged.

The importance of choosing P and Q is seen by considering the computation involved in elimination. Its basic calculations are defined by the following recursion:

$$\beta^{(1)} = PBQ^T$$

$$\beta_{ij}^{(k+1)} = \beta_{ij}^{(k)} - \beta_{ik}^{(k)}\beta_{kj}^{(k)}/\beta_{kk}^{(k)} \quad , \quad i, j > k; \; k = 1,\ldots, m-1$$

of which L and U are a by-product:

$$L_{ij} = \beta_{ij}^{(j)} \quad , \quad i \geq j$$

$$U_{ij} = \beta_{ij}^{(i)}/\beta_{ii}^{(i)} \quad , \quad i \leq j$$

The crucial values are the "pivots" $\beta_{kk}^{(k)}$: an LU factorization exists if and only if all pivots are nonzero. Moreover, elimination is numerically stable only if all pivots are sufficiently large in magnitude, both absolutely and relative to $\beta_{ik}^{(k)}$ or $\beta_{kj}^{(k)}$. As a consequence, practical Gaussian elimination looks for permutations P and Q such that PBQ^T has an acceptably large series of pivots.

Sometimes it is convenient to look at elimination in a slightly different way. B is not permuted; but the kth pivot may be any non-zero element $\beta^{(k)}_{p(k),q(k)}$ of $\beta^{(k)}$. The recursion functions for elimination then take on the following general form:

$$\beta^{(1)} = B$$

$$\beta^{(k+1)}_{p(i),q(j)} = \beta^{(k)}_{p(i),q(j)} - \beta^{(k)}_{p(i),q(k)}\beta^{(k)}_{p(k),q(j)}/\beta^{(k)}_{p(k),q(k)} ,$$

$$p(i), q(j) > k; \; k = 1, \ldots, m-1$$

L and U are yielded as before:

$$L_{ij} = \beta^{(j)}_{p(i),q(j)} , \qquad p(i) \geq q(j)$$

$$U_{ij} = \beta^{(i)}_{p(i),q(j)}/\beta^{(i)}_{p(i),q(i)} , \qquad p(i) \leq q(j)$$

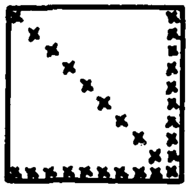
Choosing the functions p and q is called pivot selection. It is easily seen that for every p, q there are permutation matrices P, Q that produce the same L and U, and vice versa: p is just the row permutation produced by P^{-1}, and q the column permutation produced by Q^{-T}. Thus permuting B and selecting pivots from B are equivalent ways of looking at elimination, and hereafter these two points of view will be used interchangeably as convenience dictates.

Practical schemes for pivot selection are of two general types. Dynamic selection defers choosing the kth pivot until the kth step of the recursion. Preselection chooses all or part of the pivot order in advance and modifies its chosen order only if a pivot proves unacceptable. Dynamic selection is the more powerful approach, as it may examine $\beta^{(k)}$ to choose $p(k)$ and $q(k)$; but in many cases preselection can yield acceptable results more simply and cheaply.

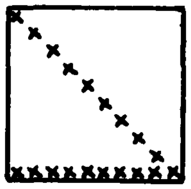
For general (non-sparse, indefinite, non-symmetric) B, the sole aim of pivot selection is to insure numerical stability. For this purpose the preferred scheme is a compromise between dynamic selection and preselection. The row pivot order, for example, is chosen in advance; the kth pivot column is then chosen dynamically to maximize $|\beta^{(k)}_{p(k),j}|$ over the remaining columns j. Alternatively, the column pivot order is chosen in advance and the kth pivot row maximizes $|\beta^{(k)}_{i,q(k)}|$ over i. (Full dynamic selection might maximize $|\beta^{(k)}_{ij}|$ over all remaining i and j. It would be more costly, however, and would usually gain little in stability.)

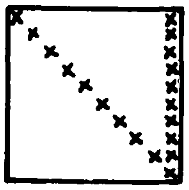
Principles of sparse elimination

The fundamental observation of sparse Gaussian elimination is the following: sparsity of the factors L and U is highly sensitive to the choice of pivots in B. A well-known example is the matrix whose nonzeroes are arranged like this:

B 

If pivots are taken in order down the diagonal, L and U retain all of the sparsity of B:

L 

U 

But if pivots run <u>up</u> the diagonal, L and U are very likely 100% dense! Experiments with less structured matrices of various sorts [11,13,14] have shown that proportional sparsity of L and U can also vary widely with pivot selection. The central task of sparse Gaussian elimination, therefore, is choosing pivots to preserve sparsity as well as numerical stability.

The relation of pivot selection to sparsity is best explained by referring again to the recursion equation:

$$\beta_{ij}^{(k+1)} = \beta_{ij}^{(k)} - \beta_{ik}^{(k)}\beta_{kj}^{(k)}/\beta_{kk}^{(k)}$$

If $\beta_{ij}^{(k)} = 0$, then $\beta_{ij}^{(k+1)} = 0$ only if $\beta_{ik}^{(k)} = 0$ or $\beta_{kj}^{(k)} = 0$; otherwise, $\beta_{ij}^{(k+1)}$ "fills in" with a non-zero element. On the other hand, if $\beta_{ij}^{(k)} \neq 0$ then also $\beta_{ij}^{(k+1)} \neq 0$ unless, fortuitously, $\beta_{ij}^{(k)} = \beta_{ik}^{(k)}\beta_{kj}^{(k)}/\beta_{kk}^{(k)}$; in the latter case $\beta_{ij}^{(k+1)}$ is "cancelled" back to zero. In practice cancellation is fairly rare, and so it is legitimate to consider only the "worst case" in which there is fill-in but no cancellation. The submarices $\beta^{(k)}$ then become progressively denser. And, since $L_{ij} = \beta_{ij}^{(j)}$ and $U_{ij} = \beta_{ij}^{(i)}/B_{ii}^{(i)}$, all nonzero fill-in is eventually reflected in L or U. In the best possible case, there is no fill-in and L, U have be same nonzeroes as the upper and lower triangles of B. At worst, every element $\beta_{ij}^{(k)}$ eventually fills in for some k, and L, U are 100% dense. (The example above shows both best and worst, for different pivot choices.)

Preserving the sparsity of L and U is thus essentially a job of controlling fill-in. Other methods of direct solution--for example, Gauss-Jordan elimination and orthogonal factorization by Givens or Householder transformations--also suffer from fill-in. Of all popular methods, Gaussian elimination controls fill-in the best, and hence it is preferred for solving large sparse systems. Under suitable assumptions, Gaussian elimination can also be shown to require minimal storage among all algorithms that compute L and U [6].

Minimizing fill-in in Gaussian elimination implies a number of economies. Certainly, it means that a minimal amount of storage is required to hold the nonzeroes of L and U. It also generally means that L and U can be computed faster, since only additions and multiplications with nonzeroes must be carried out. Solution of $L(Ux) = b$ by back-substitution is likewise more efficient when L, U are sparser.

(Minimizing fill-in is not equivalent to minimizing additions and multiplications, and there may be no pivot order that minimizes both. However, for sufficiently large and sparse B the difference between minimal fill-in and minimal computation is negligible [7].)

When B has some structure, it may also be worth controlling fill-in so that the structure is preserved in L and U. Often structure can be exploited to further reduce the computation involved in elimination and in back substitution.

It remains to say how to find a pivot order that minimizes fill-in. Unfortunately, for a general B there is no known algorithm that minimizes fill-in without essentially looking at all $m!$ possible pivot orders. Moreover, the minimal-fill-in problem has been shown to be NP-complete [43], so there is little hope of finding an efficient algorithm for it.

Consequently, practical methods for sparse elimination use heuristic techniques to keep fill-in low, if not necessarily minimal. Successful techniques are of two kinds. Structural techniques first permute B to reveal some special structure, then preselect a pivot order known to be good for that structure. Techniques of local minimization (or merit techniques) dynamically select each pivot to minimize fill-in or some related function of $B^{(k)}$. Both of these approaches are described further below, after which the two are briefly compared.

Regardless of technique, some compromise with numerical stability must be made in order to promote sparsity. In practice a compromise is achieved by weakening the standard for an acceptable pivot element: rather than being the element of largest magnitude in its row or column of $\beta^{(k)}$, the pivot need only be "not too small" relative to the element of largest magnitude. More precisely, $\beta_{kk}^{(k)}$ is acceptable as a pivot element if it exceeds some absolute tolerance ε and if either

$$\beta_{kk}^{(k)} \geq \delta\beta_{kj}^{(k)} , \qquad \text{for all } j > k$$

or

$$\beta_{kk}^{(k)} \geq \delta\beta_{ik}^{(k)} , \qquad \text{for all } i > k$$

Values of δ from 10^{-1} to 10^{-4} have been found to afford reasonable stability and sparsity [13,15,54].

Structural techniques

Sparse elimination was first applied to particularly strong and regular structures arising from models of various physical systems. In these cases a "natural" best pivot order could often be found by inspection. For example, a 1961 survey by Parter [40] analyzes tridiagonal, five-diagonal and various "looped" systems, as well as systems that arise from Laplace and biharmonic equations. More recently Varah [55,56] and Keller [35] have investigated pivot orders for block-tridiagonal and special staircase systems that arise in solution of the two-point boundary-value problem.

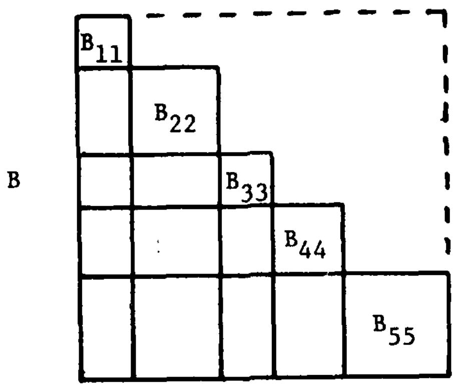
Many other applications yield systems that have no such strong structure. Yet these systems often do have a weaker, more general structure that can be exploited to choose good (if not best) pivots and to save storage and computation. Structures of this sort are usually not evident from inspection; they must be found through systematic analysis of the matrix B by one or more specially-designed algorithms. To be practical these algorithms must themselves be programmed for the computer, so that they may be run just prior to the elimination algorithm.

Such algorithms have been proposed for several kinds of structures. For example, Alway and Martin [1] considered looking for a banded form of a matrix; Weil and Kettler [58] proposed a way of permuting matrices to block-angular form.

One particular general structure--the block-triangular form--is by far the best known and most widely applied. To define block-triangularity formally, one specifies a partition $(m_1,\ldots,m_t)$, $\sum_1^t m_i = m$, of both the rows and columns of B; this induces a partition of B into m^2 blocks B_{ij}:

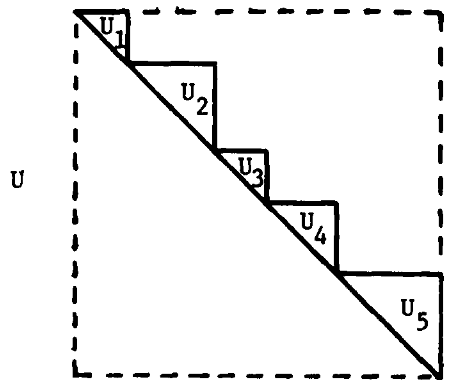
$$B \quad \begin{array}{|c|c|c|c|c|}\hline B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\ \hline B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\ \hline B_{31} & B_{32} & B_{33} & B_{34} & B_{35} \\ \hline B_{41} & B_{42} & B_{43} & B_{44} & B_{45} \\ \hline B_{51} & B_{52} & B_{53} & B_{54} & B_{55} \\ \hline \end{array}$$

Partitioned in this way, B is (lower) block-triangular if $i < j$ implies $B_{ij} = 0$:



When B is nonsingular, so is each of the square diagonal blocks B_{ii}.

Given a block-triangular partition of B, there is a natural good way to choose pivots: first pick m_1 pivots in B_{11}, then m_2 pivots in B_{22}, and so forth through B_{tt}. It is easily seen that pivots in B_{ii} will then cause fill-in within blocks B_{ii}, $B_{i+1,i}$, ..., B_{ti} only, and that the total fill-in of B will thereby tend to be limited. In fact elimination of B_{ii} has no effect at all upon columns in subsequent partitions, so that there is a reduction in computation as well. Moreover, the resulting factor U has a block-diagonal form:



that makes the back-substitution particularly easy. Better yet, it may be possible to choose the partition so that, for some values of i, $m_i = 1$. Then there is just one "triangle column" in partition i; this column appears unchanged in L, while the corresponding column of U is just a unit vector.

(There is also a block-form of the elimination algorithm [52] that just factors $B_{ii} = L_i U_i$ for each diagonal block. This may be even sparser and more efficient than standard elimination [17,23], but it does require that B be available during back-substitution.)

In light of the above observations, it may pay to look for a "best" block-triangular permutation of B. Certainly a best permutation should be "irreducible": no diagonal blcok B_{ii} should itself be permutable to block-triangular form with more than one block. A best permutation should also have as many diagonal blocks as possible. Fortunately, B always has a permutation best in both of these senses, and essentially unique: if any other permutation is just as good, it must have the same diagonal blocks (possibly in a different order), and the same rows and columns in each block. Thus it is proper to speak of <u>the</u> block-triangular reduction of B. (Existence of this reduction was first shown by Dulmage and Mendelsohn [18,19], as a by-product of their study of structures of bipartite graphs, and it has since been demonstrated in other ways.)

The block-triangular form of any nonsingular B may be found systematically in two steps:

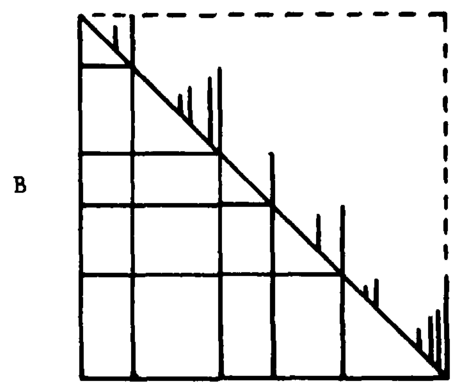
(1) Find a permutation P so that BP has no zeroes on its diagonal. This step is called finding a <u>transversal</u> of B, and it can be accomplished for any nonsingular B.

(2) Find a permutation Q so that $Q^T(BP)Q$ is block-triangular with as many diagonal blocks as possible. This is equivalent to the problem of finding and ordering the <u>strong components</u> of the graph of BP (or the irreducible classes of the Markov chain whose transitions are defined by the nonzeroes of BP).

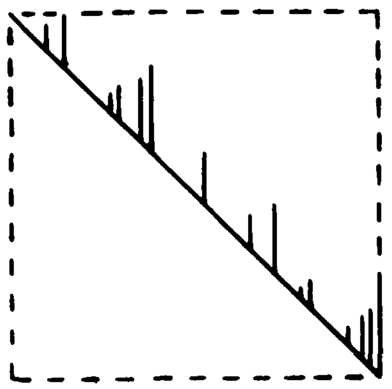
Dulmage and Mendelsohn [20,21] showed that the resulting matrix, $Q^TB(PQ)$, must be the block-triangular reduction of B. Step (1) requires only a column permutation, and step (2) only a symmetric permutation, and both are inherently simpler than finding a general permutation of the rows and columns of B. Moreover, both steps involve well-known problems for which there are efficient algorithms. Duff [12] reviews transversal algorithms; Duff and Reid [16] and Gustavson [26] describe implementations of the method of Tarjan [49] for finding strong components.

The computational effort involved in each algorithm has been obeserved to be a roughly linear function of the number of rows of B and of its number of nonzero elements. Since the comparable effort in Gaussian elimination is at best a quadratic function, block-triangular reduction is relatively cheap for large systems [13,17].

There remains the problem of choosing pivots within each diagonal block B_{ii}, whenever $m_i > 1$. This, too, is amenable to structural techniques. A number of ideas have been put forth [34,48,53]; the only approach that is widely used, however, is due to Hellerman and Rarick [27,28]. It is based on the observation that, even within a block B_{ii}, any triangle column--one that has no elements above the diagonal--appears unchanged in L and cannot fill in. All fill-in is confined to "spike" columns that have elements above the diagonal. The idea, therefore, is to permute each B_{ii} so that most columns are triangle columns, while a relative few are spikes. B then has a "bump and spike" structure such as this:



and, as before, B's structure is inherited by the factor U:

U 

Only the spike columns in U need be stored, since all others are unit vectors.

No effective algorithm is known for finding a best spike structure in any useful sense. It is therefore necessary to rely on heuristics that seem to find a good structure. The standard for this purpose is Hellerman and Rarick's Preassigned Pivot Procedure (P3). As implemented in various linear programming codes, it is regularly observed to reduce the proportion of spikes in a bump to 10-20%.

P3 does not have the inherent efficiency of the algorithms for transversal or strong components. However, P3 is applied only to the blocks B_{ii}, not to the whole matrix B. Hence P3's efficiency is acceptable provided that B's block-triangular reduction has no very large diagonal blocks. This is often--but not always--the case in practice, as the staircase examples of Sections 3 and 5 will show.

Special care must be taken to maintain the bump and spike structure in the face of numerical instability. Any rearrangement of the rows will tend to destroy the structure; consequently a too-small pivot must be avoided only through changes in the column order. Practical implementations employ the following strategy: if the kth pivot turns out to be unacceptably small, exchange column k with a later column s such that $\beta_{ks}^{(k)} > \varepsilon$. To promote stability, s is usually also chosen so that $B_{ks}^{(k)} \geq \delta B_{kj}^{(k)}$ for $j > k$.

It is easily shown that $\beta_{kj}^{(k)} = 0$ if j is a triangle column or if j and k lie in different bumps. Thus column k may only be exchanged with a spike column s in the same bump. Such an exchange is called a "triangle swap" or a "spike swap" according to whether k is a triangle or spike column. A triangle swap may increase the number of spikes by one, whereas a spike swap leaves the number of spikes unchanged. (Triangle swaps may be safely avoided, however, provided that the ratio of largest to smallest element in any column of the LP does not exceed $1/\delta$.)

When the pivot order is determined by P3, it sometimes happens that a spike's pivot element is identically zero. This is generally not a sign of numerical instability, but is due rather to an unfortunate choice of spike order. A suitable spike swap will remedy the situation.

Techniques of local minimization

Markowitz [37] first suggested that a sparse pivot order might be found by minimizing some criterion at each pivot. Methods of this sort consider, at the kth elimination step, all nonzero elements of $\beta^{(k)}$ that are large enough to serve as pivots; they associate with each such element an appropriate "merit function," and take as the kth pivot an element that minimizes this function. Thus each pivot is "locally" optimal in some sense, and it can be hoped that the resulting pivot order is close to globally optimal.

Two obvious candidates for merit function are total fill-in due to the kth pivot, and total arithmetic operations in the kth step of the elimination. It was thought at first that minimizing either of these functions locally might necessarily minimize it globally, but simple counterexamples were demonstrated by Douglas [10] and Bertelè and Brioschi [5].

Computing total fill-in or total arithmetic for a pivot in $\beta^{(k)}$ is a fairly time-consuming procedure. Consequently, there have been proposals for simpler merit functions that rely only on $r_i^{(k)}$, the number of nonzeroes in row i of $\beta^{(k)}$, and $c_j^{(k)}$, the number of nonzeroes in column j of $\beta^{(k)}$. Two typical functions of this sort are defined as follows:

- Minimize $(r_i^{(k)} - 1)(c_j^{(k)} - 1)$ over all potential pivots $\beta_{ij}^{(k)}$. This function, the one proposed originally by Markowitz, is a crude upper bound on the fill-in due to a pivot on $\beta_{ij}^{(k)}$. It is also approximately the number of multiplications and divisions required by the pivot.
- Pick a pivot column j^* that minimizes $c_j^{(k)}$; then pick a pivot row i^* that minimizes $r_i^{(k)}$ over all potential pivots $\beta_{ij^*}^{(k)}$. Or, first pick i^* to minimize $r_i^{(k)}$, then j^* to minimize $c_j^{(k)}$ over all potential pivots $\beta_{i^*j}^{(k)}$.

Experiments with these and similar criteria have been reported by Tewarson [50], Dantzig et al. [9], Curtis and Reid [8], and Duff and Reid [14]. The total-fill-in and total-arithmetic merit functions have generally performed better than the simpler functions, but only moderately so; no function has been consistently the best. Markowitz' original merit function offers perhaps the best combination of effectiveness and simplicity of those studied.

In a few experiments local minimization has produced less fill-in than the bump-and-spike structural techniques [23]; however, there are no definitive comparisons. Certainly local minimization has the advantage of being able to take any nonzero of $\beta^{(k)}$ as a pivot, without regard to preserving some structure; on the other hand, it cannot use structure to advantage. Its greatest drawback is its need to know where all the nonzeroes are in $\beta^{(k)}$, while efficient structural techniques [22] only explicitly compute column k of $\beta^{(k)}$. For local minimization $\beta^{(k)}$ may share storage with L and U--since $\beta^{(k)}$ shrinks as L and U are determined--but such an arrangement requires a "cushion" of unused storage space, plus special storage-management routines [17].

If the sparsity pattern of $\beta^{(k)}$ is not available, it is still possible to do a sort of local minimization based on r_i and c_j, the original numbers of nonzeroes in the rows and columns of B. Typically the columns are ordered in advance of the elimination, according to some merit function; as the elimination progresses, a pivot row is chosen for each column according to a second function. (Of course, the roles of rows and columns may be interchanged.) Duff and Reid [14] found that taking just c_j as the column merit function, and r_i and the row merit function, yielded results about as good as other more complex functions proposed by Tewarson [51] and others. However, they also determined that substituting $r_i^{(k)}$ for r_i gave generally better results, and that full use of $r_i^{(k)}$ and $c_j^{(k)}$ worked best of all.

Another way to avoid carrying the full sparsity pattern of $\beta^{(k)}$ is to combine local minimization with a structural technique. For example, B may first be permuted to block-triangular form, then local minimization may be applied to choose pivots in each diagonal block. Gay [23] and Duff and Reid [17] report experiments with this hybrid technique. Other hybrids are appropriate to staircase elimination as shown in Section 6 of this paper.

Considerations in linear programming

Computer codes for linear programming use Gaussian elimination to compute an LU factorization of the basis matrix. Since the basis changes by just one column at each iteration, it is essential to be able to update L and U at most iterations; a fresh elimination is carried out only occasionally (typically every 50–100 iterations) when the updated representations of L and U become too large or inaccurate.

Methods for updating L and U are based on the ideas of Bartels and Golub [2,3], but differ considerably in details. As a result, they favor quite different approaches to sparse elimination.

Saunders' updating scheme [45,46] is designed to allow L and most of U to be stored sequentially. It is predicated upon a favorable bump and spike structure, and it exploits the fact that triangle columns of B give rise to trivial unit vectors in U. Consequently, it favors the structural techniques of sparse elimination.

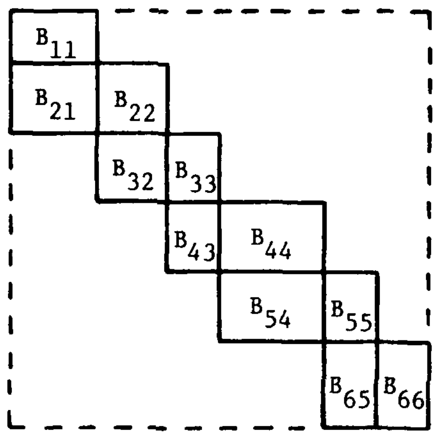
Reid's updating scheme [42] maintains sparser factors at the cost of random access to U. It works best when the bulk of nonzeroes is in U, and so prefers a local-minimization technique adjusted to favor fill-in within U. It is not so suitable to the usual bump-and-spike technique, which puts most non-zeroes in L; but it might profitably use an upper block-triangular structure that arises from applying the same techniques to B^T.

These observations suggest that no technique of sparse elimination can be considered best on the bases of fill-in and computational cost alone. It will probably always be necessary to consider different techniques for different situations.

2. STAIRCASE SYSTEMS

Staircase matrices

A "staircase" matrix is one whose nonzeroes are confined to certain submatrices centered roughly on and just off the diagonal--as, for example:

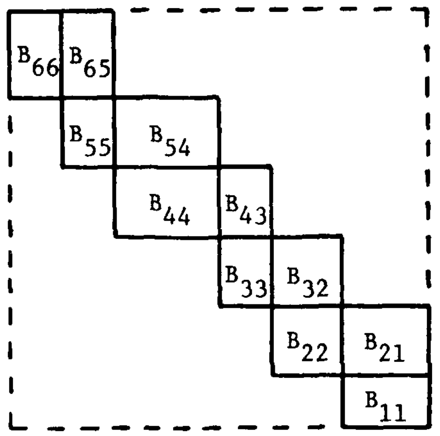


More precisely, one partitions the rows of an $m \times m$ matrix B into t disjoint subsets, and the columns into t (generally different) disjoint subsets, so that the matrix is partitioned into t^2 submatrices, or "blocks":

$$B_{ij}, \qquad i = 1,\ldots, t;\ j = 1,\ldots, t$$

B is <u>lower staircase</u> (as above) if $B_{ij} = 0$ except for $i = j$ and $i = j+1$. B is <u>upper staircase</u> if $B_{ij} = 0$ except for $i = j$ and $i = j-1$.

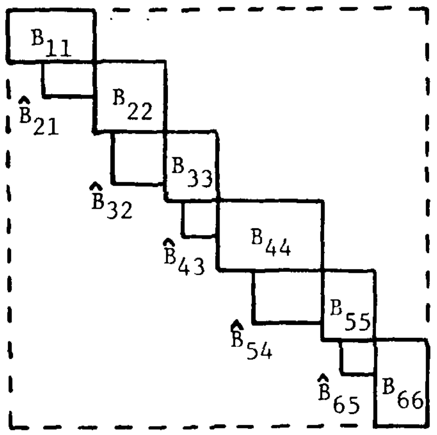
Any lower-staircase matrix may be permuted to upper-staircase form, by reversing the order of the partitions:



Likewise, any upper staircase may be permuted to lower-staircase form. The two forms are thus equivalent, and either may be considered without loss of generality. This paper adopts the lower-staircase form, and henceforth a lower staircase matrix will be called just a "staircase matrix".

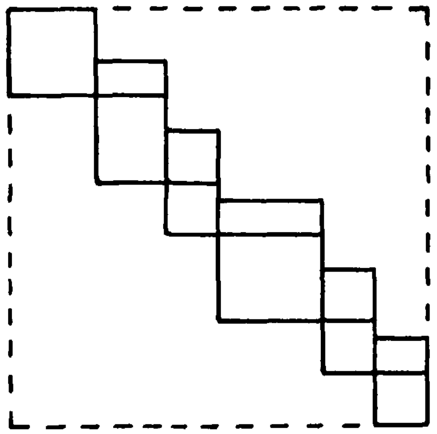
If a row of the ith partition is entirely zero within B_{ii}, that row may be moved back to the (i-1)st partition without loss of the staircase structure. Similarly, a column of the jth subset that is all-zero within B_{jj} may be moved to the (j+1)st partition. A nonsingular staircase B may thus be permuted so that its diagonal blocks B_{kk} have no all-zero rows or columns. B is then said to be in <u>standard</u> staircase form.

The off-diagonal blocks $B_{k+1,k}$ may have all-zero rows or columns whether B is in standard form or not. It is revealing to further permute the rows of the ith partition so that the nonzero rows of $B_{i,i-1}$ come first, and to permute the columns of the jth partition so that the nonzero columns of $B_{j+1,j}$ come last. Then B has the following <u>reduced</u> form:



$\hat{B}_{k+1,k}$ is just the submatrix formed from the nonzero rows and columns of $B_{k+1,k}$. The rows and columns of B that have nonzeroes in $\hat{B}_{k+1,k}$ are the <u>linking</u> rows and columns between partitions k and $k+1$.

If the linking rows of every partition k are moved back to partition $k-1$, then B gains an alternative <u>row-upper-staircase</u> form:



Moving the linking columns of partition k to partition k+1 gives a different, column-upper-staircase form. Thus any one permutation of a staircase matrix in reduced standard form embodies three staircases: lower, row-upper, and column-upper. Each corresponds to a different choice of partitions.

The staircase partitioning of B is in no way unique. Even confined to standard form, the choice is considerable; for example, either the row-upper or column-upper variant may be permuted to a lower staircase and put in its own standard form, which is generally different from the original. It is to be expected that staircase matrices arising in applications will have certain natural standard-form partitions that are best to work with, but there is no obvious way to choose the "best" staircase partition in general.

There may be other candidates for a "standard" staircase form as well. This paper's definition of reduced standard form was chosen for two reasons. First, it seems to correspond to an intuitive way of thinking about staircase systems: the diagonal blocks are full, separate sets of equations, linked together by the smaller off-diagonal blocks. Second, a single permutation of B suffices to display the reduced standard form and its upper-staircase variants. Methods of sparse Gaussian elimination also seek a permutation of B, and this staircase permutation offers a good starting point as subsequent sections will show.

The number of rows in partition i will be denoted m_i, and the number of columns in partition j will be n_j; analogously, the numbers of linking rows and columns will be $\hat{m}_i$ and $\hat{n}_j$. The number of rows in partition i of the row-upper form will be m^i, and the number of columns in partition j of the column-upper form will be n^j. Necessarily, $\sum m_i = \sum m^i = \sum n_j = \sum n^j = m$, and $\hat{m}_i \leq m_i$, $\hat{n}_j \leq n_j$. By definition,

$$\begin{array}{lll} m^1 = m_1 + \hat{m}_2 , & n^1 = n_1 - \hat{n}_1 , & \\ m^i = m_i + \hat{m}_{i+1} - \hat{m}_i , & n^j = n_j - \hat{n}_j + \hat{n}_{j-1} , & i,j = 2,\dots, t-1 \\ m^t = m_t - \hat{m}_t , & n^t = n_t + \hat{n}_{t-1} . & \end{array}$$

B_{kk} is $m_k \times n_k$, $B_{k+1,k}$ is $m_{k+1} \times n_k$, and $\hat{B}_{k+1,k}$ is $\hat{m}_{k+1} \times \hat{n}_k$.

Staircase systems

Staircase matrices B arise from staircase systems $Bx = b$. In the majority of practical staircase systems, the staircase partitions represent successive periods of time. Hence "period" means the same as "partition" in describing staircase structures.

Frequently the source of a staircase system is a "multi-period" or "time-staged" linear model of a physical or economic process. For example, an economic planning model might minimize costs over all periods, subject to constraints governing each period and its relation to the next period. If the minimum is found by some variant of the simplex method, there may well be in excess of 2m iterations, each involving solution of a staircase system with a different B.

Staircase systems are also naturally associated with linear control. Indeed, a simple discrete dynamic model with state vectors x_i,

$$x_1 = b_1$$

$$x_{i+1} = A_i x_i + b_i, \qquad i = 1,\dots, t$$

is readily seen to be a staircase system. General models of optimal control give rise to more complicated systems. Typically a control model minimizes a function of state vectors x_i and control vectors u_i, subject to dynamic equations,

$$C_i x_{i+1} = A_i^{(1)} x_i + D_i^{(1)} u_i + b_i^{(1)} , \qquad i = 1,\dots, t$$

and control constraints,

$$0 = A_i^{(2)} x_i + D_i^{(2)} u_i + b_i^{(2)} , \qquad i = 1,\ldots, t+1$$

Together, these equations form a non-square staircase system; again, if the minimization is carried out by some simplex method, there are square staircase systems to be solved at each iteration. When the resulting staircase matrices are put in reduced form, the linking rows correspond to dynamic equations, and the non-linking rows to control constraints; the linking columns correspond to state variables, and the non-linking columns to control variables.

Higher-order staircases

A more general approach says that a staircase matrix is of order r if $B_{ij} = 0$ except for $i = j, j+1, \ldots, j+r$. The preceding subsections thus characterized matrices, and systems, of order one. Higher-order staircase systems are not uncommon in complex applications (for example, modeling energy systems [39]). They arise also from control models that have rth-order dynamic equations.

This paper is predominantly concerned with first-order staircase systems: these have the most specialized structure and, consequently, are most amenable to special techniques. Still, some techniques are applicable to higher-order staircases as well, with appropriate modifications that will be pointed out as the exposition proceeds. For brevity, however, the adjective "first-order" will usually be dropped.

Higher order systems can also be made into first-order ones, in either of two ways. First, an rth-order system can be <u>transformed</u> to a first-order one by adding certain variables and equations, yielding a larger system that has the same number of periods. Second, every r periods of an rth-order system may simply be <u>aggregated</u> as one period, resulting in a first-order system of the same size having only about t/r periods. The first method is most practical when the system has only a few nonzeroes below the staircase to begin with, while the second may be feasible when the number of periods is large relative to r.

"Balanced" staircases

As defined, each period of a staircase matrix may have any number of rows and columns, subject to $\sum m_i = \sum n_j = m$; and, of these, any number may be linking rows and columns, subject to $\hat{m}_i \leq m_i$ and $\hat{n}_j \leq n_j$. But if $Bx = b$ is to have a solution for arbitrary b, then B must be a nonsingular staircase matrix. Nonsingularity imposes much stronger constraints upon the shape of the staircase. Essentially, a nonsingular staircase matrix must be well "balanced": there cannot be too great a difference between the numbers of rows and columns, either in individual periods or cumulatively.

The simplest balance relations are derived as follows. Given that B is nonsingular, the columns in any period k must be independent. But the n_k columns of period k have nonzero elements only in blocks

B_{kk} and $B_{k+1,k}$, and so they can be independent only if $n_k \leq m_k + m_{k+1}$. With special consideration of the last period, this shows that

$$n_k \leq m_k + m_{k+1}, \qquad k = 1,\ldots, t-1$$
$$n_t \leq m_t$$

The rows of a nonsingular matrix must also be independent. Hence, by a similar argument,

$$m_1 \leq n_1$$
$$m_k \leq n_{k-1} + n_k, \qquad k = 2,\ldots, t$$

Putting these relations together, one gets

$$0 \leq n_1 - m_1 \leq m_2$$
$$-n_{k-1} \leq n_k - m_k \leq m_{k+1}, \qquad k = 2,\ldots, t-1$$
$$-n_{t-1} \leq n_t - m_t \leq 0$$

The difference $n_k - m_k$ is the imbalance between rows and columns of period k. Taken together, these inequalities tend to rule out a very large imbalance in any period.

A stronger set of inequalities can be derived by also considering the numbers of linking rows and columns in the reduced form of B. For example, the columns of period k really have nonzeroes only in the $m_k + \hat{m}_{k+1}$ rows of blocks B_{kk} and $\hat{B}_{k+1,k}$. Thus

$$n_k \le m_k + \hat{m}_{k+1} , \qquad k = 1,\ldots, t-1$$

$$n_t \le m_t$$

Similar reasoning applies to the rows of each period to give

$$m_1 \le n_1$$

$$m_k \le \hat{n}_{k-1} + n_k , \qquad k = 2,\ldots, t$$

Further, within each period the $n_k - \hat{n}_k$ non-linking columns must be independent. These columns have nonzeroes only within B_{kk}, and so

$$n_k - \hat{n}_k \le m_k , \qquad k = 1,\ldots, t-1$$

Analogously, the $m_k - \hat{m}_k$ non-linking rows in period k must be independent, so that

$$m_k - \hat{m}_k \le n_k , \qquad k = 2,\ldots, t$$

Combining these four sets of relationships,

$$0 \le n_1 - m_1 \le \min(\hat{m}_2,\hat{n}_1)$$

$$-\min(\hat{m}_k,\hat{n}_{k-1}) \le n_k - m_k \le \min(\hat{m}_{k+1},\hat{n}_k) , \qquad k = 2,\ldots, t-1$$

$$-\min(\hat{m}_t,\hat{n}_{t-1}) \le n_t - m_t \le 0$$

In brief, the imbalance of period k is bounded by the smaller dimension of $\hat{B}_{k,k-1}$ and the smaller dimension of $\hat{B}_{k+1,k}$. If $\hat{B}_{k+1,k}$ is small for all k--so that $\hat{m}_k << m_k$ or $\hat{n}_k << n_k$--then these bounds are especially severe. Hence "weakly linked" staircases must be nearly balanced.

All of the above analysis applies equally well to any <u>series</u> of periods k, k+1, ..., ℓ: the rows or columns of periods k through ℓ must be independent, and so various inequalities must hold. As a consequence, the constraints derived above for one-period balance can be generalized to <u>cumulative</u> balances over periods k through ℓ:

$$0 \le \sum_1^\ell (n_i - m_i) \le \min(\hat{m}_{\ell+1}, \hat{n}_\ell) , \quad \ell = 1,\ldots, t-1 \quad (1)$$

$$-\min(\hat{m}_k, \hat{n}_{k-1}) \le \sum_k^\ell (n_i - m_i) \le \min(\hat{m}_{\ell+1}, \hat{n}_\ell), \quad k,\ell = 2,\ldots, t-1 \quad (2)$$

$$-\min(\hat{m}_k, \hat{n}_{k-1}) \le \sum_k^t (n_i - m_i) \le 0 , \quad k = 2,\ldots, t \quad (3)$$

Of course, $\sum_1^t (n_i - m_i) = 0$ since the matrix B is square. If constraints (1) above are subtracted from this equality, they yield constraints (3), and vice-versa; hence (1) and (3) are equivalent.

The constraints for cumulative balance are quite strong, especially when the number of periods t is fairly large. Notably, they imply that the imbalance $n_i - m_i$ cannot stay positive or negative for too many consecutive periods; and, the greater the imbalance is on average, the more often it must fluctuate between positive and negative.

Square sub-staircases

It is revealing to rewrite the inequalities (1) in the following way:

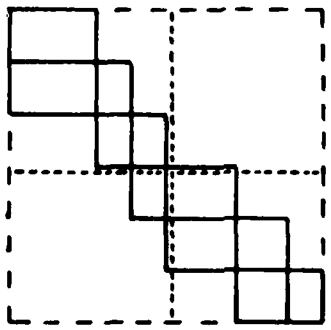
$$\textstyle\sum_1^\ell (n_i - m_i) \geq 0 \quad \Rightarrow \sum_1^\ell n_i \geq \sum_1^\ell m_i \tag{4}$$

$$\textstyle\sum_1^\ell (n_i - m_i) \leq \hat{m}_{\ell+1} \Rightarrow \sum_1^\ell n_i \leq \sum_1^\ell m^i \tag{5}$$

$$\textstyle\sum_1^\ell (n_i - m_i) \leq \hat{n}_\ell \quad \Rightarrow \sum_1^\ell n^i \leq \sum_1^\ell m_i \tag{6}$$

In words, the first ℓ periods of the lower staircase cannot have more rows than columns, while the first ℓ periods of either associated upper staircase (as defined above) cannot have more columns than rows.

The above relations are equalities when $\ell = t$, since the matrix is square. It can also happen that equality is achieved for some $\ell < t$. For example, if $\sum_1^\ell m_i = \sum_1^\ell n_i$, the matrix must look something like this:



$$\textstyle\sum_1^3 m_i = \sum_1^3 n_i$$

The rows and columns of periods 1 through ℓ form a square sub-staircase, as do the rows and columns of periods $\ell+1$ through t; they are linked only by nonzero elements in the off-diagonal block $\hat{B}_{\ell+1,\ell}$. In a similar way, an equality $\sum_1^\ell n_i = \sum_1^\ell m^i$ implies a pair of square sub-staircases within the row-upper staircase form, and $\sum_1^\ell n^i = \sum_1^\ell m_i$ implies the same for the column-upper form.

In general, a single matrix may exhibit any or all of these three kinds of equalities, and they may hold for several values of $\ell < t$. If p such equalities hold, then the matrix decomposes into $p+1$ disjoint irreducible square sub-staircases of various kinds. (Some of these sub-staircases may be null, however, if there are no linking rows or columns in certain periods.)

Block-triangular reduction of staircase matrices

Square sub-staircases have a natural relationship to the block-triangular reduction of a staircase matrix. In fact, if the staircase is dense--that is, if there are no zeroes in the blocks B_{kk} and $\hat{B}_{k+1,k}$--then each irreducible square sub-staircase coincides with one of the blocks of the reduction. A weaker corollary of this result applies to the sparser staircases commonly encountered in applications (especially linear programming): each square sub-staircase comprises one or more blocks of the reduction.

The remainder of this section outlines proofs of the above assertions. The main proof is in two parts that correspond to the two steps of the standard method for block-triangular reduction: Proposition 1 demonstrates a transversal (a nonzero diagonal) within the dense staircase, and Proposition 2 then shows that the strong components of a dense staircase are exactly the square sub-staircases. Proposition 3 applies as much of the foregoing as possible to staircases that are not dense.

PROPOSITION 1. A dense nonsingular staircase matrix, in reduced standard form, has a nonzero diagonal.

Proof. Consider any column j of period ℓ. By definition,

$$\sum_1^{\ell-1} n_i < j < \sum_1^{\ell} n_i \tag{7}$$

But nonsingularity implies, by (4) above, that $\sum_1^{\ell-1} m_i \leq \sum_1^{\ell-1} n_i$. Hence

$$\sum_1^{\ell-1} m_i < j$$

and consequently row j must also be of period ℓ or later. There are now two cases to consider:

(a) Row j is of period ℓ. The jth diagonal element is at the intersection of a row and a column of period ℓ; hence it is an element of $B_{\ell\ell}$, and must be nonzero.

(b) Row j is from a period after ℓ. With (7), this implies that

$$\sum_1^{\ell} m_i < j \leq \sum_1^{\ell} n_j$$

But nonsingularity implies, by (5) and (6) above, that $\sum_1^\ell n^i \le \sum_1^\ell m_i$ and $\sum_1^\ell n_i \le \sum_1^\ell m^i$. Hence

$$\sum_1^\ell n^i < j \le \sum_1^\ell m^i$$

which means that j must be a linking column of period ℓ and a linking row of period $\ell+1$. The jth diagonal element is thus in $\hat{B}_{\ell+1,\ell}$, and must be nonzero. ■

PROPOSITION 2. Let B be a dense nonsingular staircase matrix, in reduced standard form. Then the diagonal blocks of B's block-triangular reduction are exactly the irreducible square sub-staircases of B.

Outline of proof: In light of Proposition 1, it suffices to show that the strong components of B correspond to the irreducible square sub-staircases. Since strong components are defined by a symmetric permutation, any strong component may be identified by its diagonal elements.

Proposition 1 implies that every element of the diagonal must fall into one of the following classes:

D_ℓ	diagonal elements in $B_{\ell\ell}$
E_ℓ	diagonal elements in $\hat{B}_{\ell+1,\ell}$

It is straightforward to verify that, if class D_ℓ is not null, all of its members are in the same strong component. The same is true of E_ℓ.

These classes have a natural ordering--D_1, E_1, D_2, E_2,..., D_{t-1}, E_{t-1}, D_t--such that the following is also straightforwardly shown: if two classes are in the same strong component, then so are all classes between them in the ordering. It follows that each strong component comprises one chain of adjacent classes: for example, the first strong component might contain D_1 through D_3, the second E_3 through D_6, and so forth. To specify the strong components, then, it suffices to say where each chain ends--or equivalently, it suffices to find every place where two adjacent classes are not in the same strong component.

When are adjacent classes in different strong components? It is again straightforward to show that this can happen only in three cases:

(a) D_ℓ contains no element on any period-ℓ linking row

(b) D_ℓ contains no element on any period-ℓ linking column

(c) E_ℓ is null

and it is easily seen that these cases are equivalent, respectively, to the following:

(A) $\sum_1^{\ell-1} m^i = \sum_1^{\ell-1} n_i$

(B) $\sum_1^{\ell} m_i = \sum_1^{\ell} n^i$

(C) $\sum_1^{\ell} m_i = \sum_1^{\ell} n_i$

Thus it can be concluded that the strong components correspond exactly to the irreducible square sub-staircases. ■

PROPOSITION 3. Let B be a nonsingular staircase matrix in reduced standard form. Then each diagonal block of B's block-triangular reduction lies wholly within one of the irreducible square sub-staircases of B.

Proof: Define a dense staircase matrix $\tilde{B}$ by filling in all zeroes within the staircase blocks of B. Clearly $\tilde{B}$ has all the same square sub-staircases as B, and by Proposition 2, $\tilde{B}$'s diagonal blocks are exactly its irreducible square sub-staircases.

Now consider deriving B from $\tilde{B}$ by substituting zeroes for certain nonzeroes in $\tilde{B}$. It is easily shown that substituting a zero for a nonzero preserves the structure of the block-triangular reduction; at most, a new zero enables one diagonal block to be split into several. Thus the diagonal blocks of B must lie wholly within the diagonal blocks of $\tilde{B}$.

Putting the above statements together, one has that B's diagonal blocks lie wholly within $\tilde{B}$'s diagonal blocks, which are exactly $\tilde{B}$'s square sub-staircases, which are the same as B's square sub-staircases; hence the Proposition has been proven. ■

3. ELIMINATION OF STAIRCASE SYSTEMS

Test problems

Staircase systems for the computational tests in this paper were derived from a variety of staircase-structured linear programs (LPs). Test runs employed versions of the MINOS LP code [38,47] specially modified to incorporate different sparse elimination techniques. As explained in Section 1, MINOS merely updated the LU factorization at most iterations; a fresh "refactorization" was performed only occasionally, typically 10-20 times in a test run. At each refactorization one or more eliminations were performed and pertinent statistics were collected; these statistics were the raw data for the results reported here.

The following staircase LPs were used in one or more tests:

NAME OF TEST LP	PERIODS	ROWS	COLUMNS	NONZERO ELEMENTS	FIRST-ORDER?
GROW15	15	301	645	5666	Yes
GROW22	22	441	946	8319	Yes
SCSD8	39	398	2750	11349	Yes
SCAGR25	25	472	500	2208	Yes
SCRS8	16	491	1169	4106	Yes
SCFXM2	8	661	914	5466	Yes
SCTAP2	10	1101	1880	13815	Yes
PILOT	9	723	2789	9291	No
BP1	7	822	1571	11414	No

All but two are first-order staircase LPs, and PILOT is nearly first-order (there are less than 100 elements outside the main staircase). BP1, by contrast, has only a weak, high-order staircase form--it was not originally formulated as staircase--and it was included to test the robustness of some of the methods. GROW15 and GROW22 are essentially twins except in number of periods; otherwise all of these LPs differ considerably in structure and proportion, as Appendix A shows in more detail. Staircase bases in the test runs thus also differed greatly from LP to LP; moreover, for some LPs the basis structure changed noticeably from refactorization to refactorization. Thus the LP bases afforded a well-varied set of test matrices for elimination.

Three separate groups of test runs are referred to in the following sections. Each test group employed a subset of the LPs listed above, and produced a particular set of statistics.

The first group of tests was designed to yield data on structural techniques for sparse elimination. At each refactorization MINOS performed two eliminations, one by a standard method and one by the staircase method developed in Section 5, and reported extensive statistics on both eliminations and on the basis structure. These statistics were averaged to produce the numbers reported below. Where the statistics were percentages--percent fill-in, for example--individual percentage figures were computed for each elimination, and these figures were then averaged. (Reported percentages are thus not based on the averaged data, but are essentially weighted percentages of the original data.)

The first test group used all of the above LPs except SCTAP2, whose bases are consistently almost triangular. Runs were started from an all-slack basis except for PILOT and BP1; consequently, some early bases had an uncharacteristically high proportion of unit vectors and were disregarded. The remaining bases were classified according to the size of the largest bumps in their block-triangular reduction. For SCAGR25, SCRS8, SCFXM2, PILOT and BP1 the bases were divided into two classes, one of larger and one of smaller bumps; these classes are designated SCAGR25+ and SCAGR25-, SCRS8+ and SCRS8-, and so forth, and separate averages are reported for each. The bases of SCSD8 were similarly divided into three classes, denoted SCSD8- (small bumps), SCSD8+ (larger bumps), and SCSD8++ (largest bumps). The bases of GROW15 and GROW22 had bumps of a fairly uniform size and so were not classified.

The second group of tests was similar to the first, but yielded data on both structural and local-min techniques for sparse elimination. Five eliminations were performed at each refactorization: two by standard methods, and three by staircase methods of Sections 5 and 6. Statistics were collected, classified, and averaged as for the first test group.

Tests in the second group used GROW15, SCSD8, SCAGR25, SCRS8 and SCFXM2; bases were divided into large-bump and small-bump classes except for GROW15. The design of MINOS favors structural techniques, and the test implementation of local-minimization techniques was fairly inefficient; as a consequence the second test group comprised a smaller set of bases than the first, and the data generally have a greater variance about the averages.

A third group of tests measured the amount of time spent in each LP subroutine. This group used all of the LPs except GROW15 and GROW22; each LP was timed twice, once using the standard structural technique for sparse elimination and once using the staircase structural technique of Section 5. (Other results from these timing runs are reported in [22].)

All runs employed a relative pivot tolerance $\delta = 10^{-4}$. This is a smaller δ than is usually used, but it gave adequate results in virtually every case. (Only four bases--one each from SCAGR25, SCRS8, SCFXM2 and PILOT--yielded unacceptable factorizations, both from structural techniques; MINOS automatically refactorized each of them with a higher δ.) A smaller δ results in fewer rejected pivots, so that generally less time is spent in factorization.

Further details of all three groups of tests are summarized in Appendix B.

Standard structural techniques

Bump-and-spike analysis has been the standard structural technique for sparse elimination in large-scale linear programming codes. As such, it has been applied successfully to a great variety of structured linear systems. How, then, does it perform on staircase structures? Experimental results, summarized in Appendix C and in Section 5 below, seem to indicate that bump-and-spike analysis may do well but sometimes does quite poorly.

The critical factor is clearly the size of the bumps in the block-triangular reduction. When the reduction yields many small bumps, the elimination proceeds quickly and fill-in is low. When there are fewer and larger bumps, however, performance degrades markedly: fill-in is greater, running time of the P3 heuristic increases steeply, and there tend to be more spike swaps due to unsatisfactory pivots. The situation is especially bad when half or more of the basis columns are in one or a few large bumps.

Such behavior is predictable: bump-and-spike techniques are predicated upon a block-triangular structure having many small bumps, and they cannot be expected to work as well when this desired structure is absent. The important thing is that--as suggested in Section 2--staircase systems may be prone to large bumps when there are few square sub-staircases. Indeed, the data for SCSD8 show a clear correlation between bump size and number of lower sub-staircases. (No clear relationship appears for the other examples, but it must be kept in mind that a system may have alternate staircase forms that go unrecognized.)

In sum, staircase systems seem especially likely to have large bumps. When they do, standard bump-and-spike techniques lose their attractiveness, and it is worth looking for alternatives.

Bump-and-spike analysis also suffers from ignorance of the staircase structure. It does find the square sub-staircases, since they each comprise one or more bumps. But, within a square sub-staircase, it tends to scramble the rows and columns without regard to period. Consequently there is little structure to L or U that might be exploited to save time in back-substitution. Experiments in [22] have shown that such savings can be significant.

Standard local-minimization techniques

Appendix D and Section 6 below show statistics from a standard local-minimization technique applied to elimination of staircase systems. Results seem encouraging: fill-in is equal to or less than that for bump-and-spike techniques, and is much less sensitive to bump size. Moreover, there is no reason to expect a gross increase in running time when a few bumps are large. Thus local-minimization techniques appear somewhat preferable to bump-and-spike structural techniques in dealing with large-bump staircase bases.

Local minimization is at a disadvantage in handling large systems, however, since it must keep track of the uneliminated matrix $\beta^{(k)}$ at every step. Staircase systems in particular are often large ones, since they are generally built from one-period models that are not small to begin with. Thus local-minimization techniques may involve undesirable amounts of extra storage or storage-management overhead.

Local minimization also suffers, like bump-and-spike analysis, from ignorance of the staircase structure. It generally scrambles all rows and columns without regard to periods, and does not even preserve square sub-staircases. Hence none of the staircase structure is passed on to L and U.

4. NATURAL PIVOT ORDERS FOR STAIRCASE SYSTEMS

Motivation

Many kinds of structured systems have "natural" pivot orders that can be proved to yield no fill-in or limited fill-in. Staircase systems are no exception. Described below are two natural pivot orders for staircase matrices: one involves a dynamic ordering of rows and columns, while the other preorders the rows and dynamically orders the columns only. These natural orders motivate the staircase pivot-selection methods of Sections 5 and 6.

In describing pivot orders it is convenient to first consider a <u>dense</u> staircase matrix that is entirely nonzero within the blocks B_{kk} and $B_{k+1,k}$ (or $\hat{B}_{k+1,k}$ if in reduced form). Strong limits to fill-in can be proved for dense staircases; these limits are then readily generalized to sparser staircases, which are the ones commonly encountered in applications (especially in linear programming).

Both of the pivot orders described here are prefigured in earlier work. A similar dynamic ordering was derived by Varah [56] for certain kinds of staircase systems. Saigal [44] and Propoi and Krivonozhko [41] independently developed a method for staircase LPs that preorders the rows and dynamically orders the columns; although they did not think in terms of elimination, their method must lead to a pivot order like the second one below. All of these authors, however, assume that sparsity within the staircase blocks can be ignored, and their methods do not extend directly to sparse staircases.

Natural pivot orders with dynamic selection

Consider first a dense staircase matrix in reduced standard form, and allow its rows and columns to be rearranged in any way to avoid undesirable pivot elements. As explained in Section 1, a nonzero element of $\beta^{(k)}$ may be accepted as the kth pivot if it is at least δ times the largest element in its column of $\beta^{(k)}$, or if it is at least δ times the largest element in its row of $\beta^{(k)}$.

In this case there need be no fill-in at all:

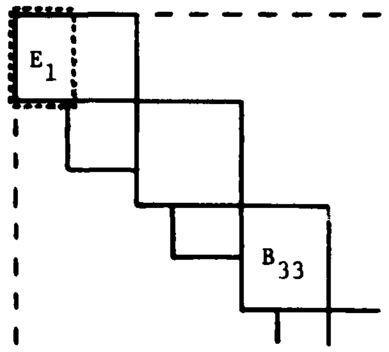
PROPOSITION 4. If B is a nonsingular dense staircase matrix in reduced standard form, then there is an acceptable pivot order for B that produces no fill-in.

Proof is by induction on the number of periods. The one-period case is trivial. The induction step considers a t-period case: the idea is to find k acceptable pivots such that (i) no fill-in occurs, and (ii) the remaining uneliminated matrix, $\beta^{(k+1)}$, is a (t-1)-period staircase matrix. By the induction hypothesis, pivots may then be found within $\beta^{(k+1)}$ to complete the elimination without fill-in.

The desired pivots are found in three steps, as follows:

(1) Consider the $m_1 \times n^1$ matrix--call it E_1--comprising the rows and non-linking columns of period 1:

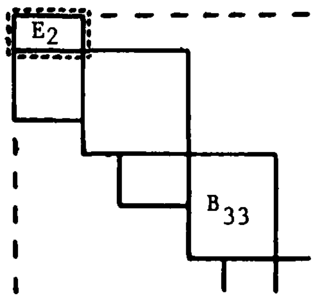
$B^{(1)}$



B nonsingular implies $m_1 \geq n^1$, and so the columns of E_1 have full rank. Consequently an acceptable pivot may be found in every column of E_1. Carry out the elimination of B with these pivots: clearly there is no fill-in, and the staircase structure is preserved. (If $n^1 = 0$ --all columns of period 1 are linking--or $m_1 = 0$, this step can be skipped.)

(2) The remaining uneliminated matrix, $B^{(n^1+1)}$, looks like this:

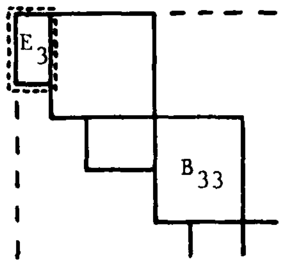
$B^{(n^1+1)}$



$\beta^{(n^1+1)}$ is the same as B except for the $(m_1-n^1)\times(n_1-n^1)$ matrix--call it E_2--comprising the remaining rows and columns of period 1. Nonsingularity of B implies $n_1 \geq m_1$, hence $(n_1-n^1) \geq (m_1-n^1)$; since $\beta^{(n^1+1)}$ must also be non-singular, it follows that the rows of E_2 must have full rank. Thus an acceptable pivot may be found in every row of E_2. Again, carry out the elimination on B with these pivots: there is no fill-in, and the staircase structure is preserved. (If $n_1 = n^1$--period 1 has no linking columns--or $m_1 = n^1$, this step may be skipped.)

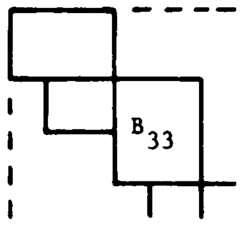
(3) The remaining uneliminated matrix, $\beta^{(m_1+1)}$, looks like this:

$\beta^{(m_1+1)}$



$\beta^{(m_1+1)}$ is the same as B except for the $(m^1-m_1) \times (n_1-m_1)$ matrix--call it E_3--comprising the remaining columns of period 1 and the linking rows of period 2. Nonsingularity of B implies $m^1 \geq n_1$, hence $(m^1-m_1) \geq (n_1-m_1)$; since $\beta^{(m_1+1)}$ must also be nonsingular, it follows that the columns of E_3 must have full rank. Thus an acceptable pivot may be found in every column of E_3. Eliminating B with these pivots, there is still no fill-in, and again the staircase structure is preserved. (This step may be skipped if $m^1 = m_1$--period 2 has no linking rows--or $m_1 = n_1$.)

At this point all rows and columns of period 1, and all but $m_1 + m_2 - n_1$ rows of period 2, have been eliminated. The remaining matrix, $\beta^{(n_1+1)}$, has this form:

$\beta^{(n_1+1)}$ 

$\beta^{(n_1+1)}$ is the same as B, except that B_{22} has been reduced to $(m_1 + m_2 - n_1) \times n_2$. Clearly, then $\beta^{(n_1+1)}$ is itself a nonsingular dense staircase matrix in reduced standard form, but with t-1 periods. Thus the proof is completed by induction, as intended. ■

The proof above demonstrates a natural pivot order: essentially, it pivots in the columns of each period before those of the next, and pivots in the rows of each period before those of the next. Pivots within each period are chosen in an order that avoids all fill-in if the staircase is dense.

When the staircase is not fully dense, some fill-in is generally unavoidable. However, an inspection of the proof of Proposition 4 shows that the natural pivot order exists whether the staircase is dense or not; the only difference in the non-dense case is that some elements within the staircase may fill in. Hence the following corollary:

PROPOSITION 5. If B is a nonsingular staircase matrix in reduced standard form, then there is an acceptable pivot order for B that produces no fill-in outside blocks B_{ii} and $\hat{B}_{i+1,i}$.

In short, there is a natural pivot order that confines all fill-in to within the staircase. It seems reasonable to expect that such a pivot order will be a good one. However, it is entirely possible that there are other, sparser pivot orders that do not confine fill-in to the staircase. (Indeed, most staircase matrices have several reduced standard staircase forms; even if the sparsest pivot order confines fill-in to one of these staircases, it may well not confine fill-in to the others.)

The natural pivot order for a given staircase matrix is hardly unique. In practice there are usually many acceptable natural pivot orders, some of them considerably better than others, and a further method is needed to choose between them. Use of a local-minimization technique for this purpose is the topic of Section 6.

Natural pivot orders with preselection of rows

Some kind of bump-and-spike structural technique might also be used to choose a natural staircase pivot order. But techniques of this sort allow only colmun swaps to avoid undesirable pivots, whereas both row and column rearrangements are generally needed to find a pivot order in the sense of Proposition 5. Hence bump-and-spike techniques require a different variety of natural pivot order--one that can be found through rearrangement of columns only.

Again, consider first a dense staircase. Fix the row order so that the rows are pivoted, as in Proposition 4, by period: all rows in period 1 are pivoted first, then all rows in period 2, and so forth through period t. Then fill-in is limited as follows:

PROPOSITION 6. Let B be a nonsingular dense staircase matrix, and fix in advance any pivot order by period of the rows of B. Then there is a corresponding acceptable column pivot order for B such that

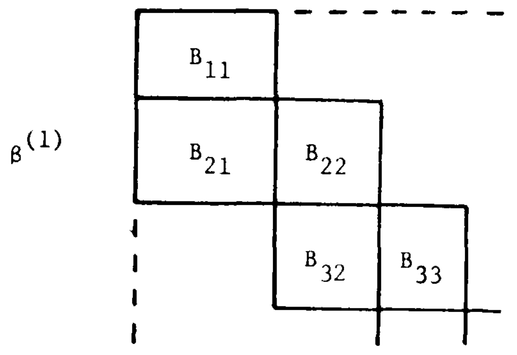
(i) Fill-in occurs in as few as $\sum_{k=1}^{t} (n_k - \text{rank } B_{kk})$ different columns.

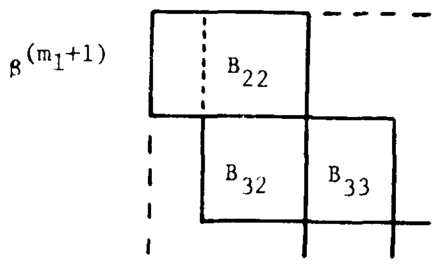
(ii) The number of elements that fill in is at most $\sum_{k=2}^{t-1} [m_{k+1} \sum_{i=1}^{k-1} (n_i - m_i)]$.

Proof is in two parts. The first part demonstrates a class of pivot orders that confine fill-in to a restricted set of columns. The second part shows how small this set can be.

(1) The rows of period 1 intersect only the columns of period 1. Consequently, a column of period 1 can be found to pivot acceptably in each row of period 1. These pivots produce no fill-in.

The remaining uneliminated matrix, $\beta^{(m_1+1)}$, looks like this:





The rows of period 2 now intersect only the leftover columns of period 1 and the columns of period 2. Consequently, a column of period 1 or 2 can be found to pivot acceptably in each row of period 2. In the worst case, each leftover period-1 column fills in along the rows of period 3; there is no fill-in elsewhere.

It is readily seen that the situation of period 2 repeats itself at each period k. In the remaining matrix, $\beta^{(m_1+\cdots+m_{k-1}+1)}$, the rows of period k intersect only the leftover columns from periods 1, ... , k-1, and the columns of period k. (The columns of periods 1, ... , k-2 did not originally intersect the period-k rows, but they have since filled in along these rows.) Consequently, a column from periods 1, ... , k can be found to pivot on each row of period k. In the worst case, each leftover column from a period before k fills in along the rows of period k+1 (except when $k = t$), but there is no fill-in elsewhere.

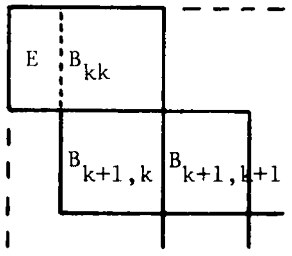
Any pivot sequence chosen in this way will maintain the fixed row order, and will confine fill-in to the "leftover" columns. Moreover, after the rows of period k-1 have been pivoted, the number of leftover columns is just the cumulative excess of columns over rows, or $\sum_{i=1}^{k-1} (n_i - m_i)$. Since these columns fill in on the rows of period k+1, the worst-case total fill-in is $\sum_{k=2}^{t-1} [m_{k+1} \sum_{i=1}^{k-1} (n_i - m_i)]$.

(2) Although the number of leftover columns after period k-1 is fixed, the total of <u>different</u> leftover columns can vary widely from one pivot choice to another. It is desirable to minimize this total, so that fill-in is confined to as few columns as possible. An exact minimum cannot be demonstrated if an acceptable pivot order is required, but it <u>is</u> possible to find a lower bound that should be very close to minimum when B is well-conditioned.

To this end, leftover columns may be characterized as follows: they are the columns from any period that pivot on rows of a later period. Thus minimizing the total of different leftover columns is equivalent to maximizing, for each k, the number of columns of period k that pivot on rows of period k.

Consider, then, the matrix $\beta^{(m_1+\cdots+m_{k-1}+1)}$ that remains after rows of periods 1, ... , k-1 have been eliminated:

$\beta^{(m_1+\ldots+m_{k-1}+1)}$



The rows of period k must pivot on columns of the submatrix $[E \;\; B_{kk}]$, where E is what remains of leftover columns from previous periods, and

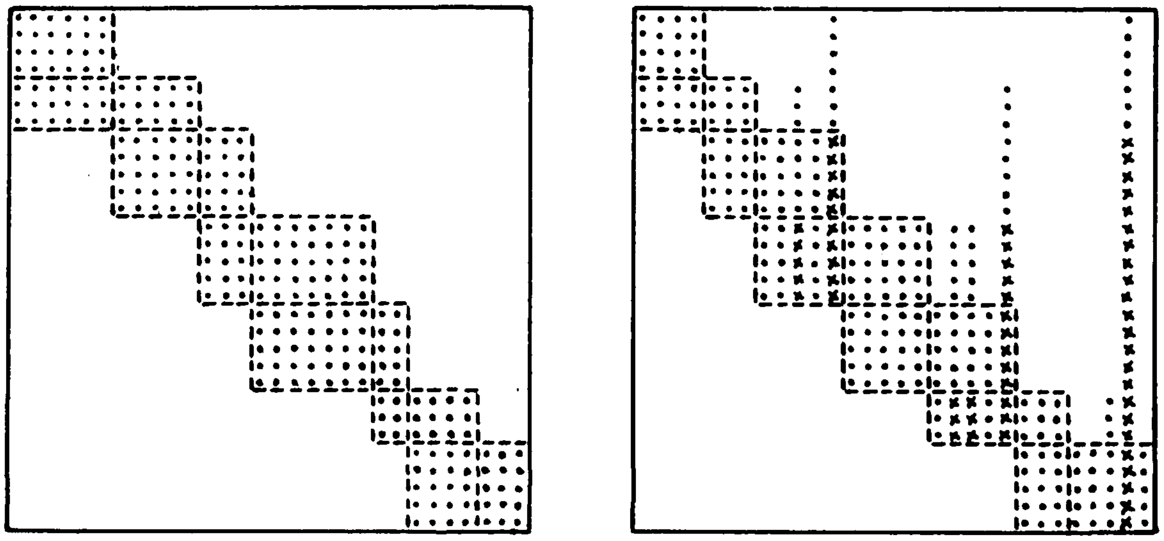
B_{kk} is the (as yet unaltered) diagonal block from period k of the staircase. Exactly m_k independent columns of $[E \quad B_{kk}]$ must be chosen to pivot on the period-k rows. The maximum number of independent columns from B_{kk}, however, is rank B_{kk}. Hence at most rank B_{kk} period-k columns can pivot on period-k rows, and it follows that the minimum number of leftover columns from period k is n_k - rank B_{kk}.

The minimum total of different leftover columns is thus $\sum_{k=1}^{t} (n_k - \text{rank } B_{kk})$. ■

Proposition 6 gives a somewhat different natural pivot order: most rows and columns are pivoted in period order--indeed, most period-k columns are pivoted on period-k rows--but, since $\sum_1^k n_i \geq \sum_1^k m_i$, some leftover columns must be pivoted on later rows. If the blocks B_{kk} are near full rank, then fill-in is confined to approximately $\sum_{k=1}^{t} \max(0, n_k - m_k)$ columns; the balance constraints of Section 2 assure that this will not be too large a number. Likewise, the total fill-in cannot be too large because the term $\sum_{i=1}^{k-1} (n_i - m_i)$ is bounded by the balance constraints.

The proof of Proposition 6 suggests a procedure for finding a natural pivot order of this kind. The procedure has t stages, one for each period; at any stage the unpivoted leftover columns are kept in a list. At stage k, find as many independent columns of B_{kk} as possible to pivot on rows of period k. Choose leftover columns from the list to pivot on any remaining period-k rows; add to the list any leftover unpivoted columns from B_{kk}.

The effect of this procedure is seen more clearly from a permutation point of view. When the columns are permuted to place all pivots on the diagonal, the staircase is arranged, for example, as follows:



• original non-zeroes ✖ fill-in

Essentially, just enough columns are moved to later periods so that the diagonal blocks B_{kk} are "squared off" and made nonsingular. In the context of bump-and-spike techniques, the B_{kk}'s are like bumps, and the columns moved to later periods are "interperiod spikes."

As previously, there is an obvious corollary for the case in which the staircase is not fully dense:

PROPOSITION 7: Let B be a nonsingular staircase matrix, and fix in advance any pivot order by period of the rows of B. Then there is a corresponding acceptable column pivot order for B such that

(i) Fill-in occurs only within blocks B_{kk} and $B_{k+1,k}$ except for as few as $\sum_{k=1}^{t} (n_k - \text{rank } B_{kk})$ columns which may fill in within blocks B_{jk}, $j > k+1$.

(ii) The fill-in within blocks B_{jk}, $j > k+1$, is at most $\sum_{k=2}^{t-1} [m_{k+1} \sum_{i=1}^{k-1} (n_i - m_i)]$ elements.

Again, there are many natural pivot orders that satisfy the requirements, and a practical method is needed for choosing a good one. An adaptation of the bump-and-spike techniques is especially appropriate for this purpose, as explained in Section 5.

5. A STRUCTURAL TECHNIQUE FOR STAIRCASE ELIMINATION

Motivation

The bump-and-spike structural technique for pivot selection, although it works well generally, is not especially efficient or effective in choosing pivots for certain staircase matrices. In light of the preceding discussion, the source of these difficulties is clear. At the heart of the problem are two assumptions of bump-and-spike analysis: first, that the block-triangular reduction of a sparse matrix yields many small bumps; and second, that the bump-and-spike structure suggests a good pivot order. Staircase matrices tend to violate both of these assumptions. Their block-triangular reductions are prone to bumps the size of square sub-staircases; and their natural pivot orders (which pivot mainly by order of period) are different from the bump-and-spike order (which scrambles periods).

These observations suggest that bump-and-spike techniques for staircase matrices cannot rely on a good square-block-triangular structure. An alternative structure is needed, and there is one readily available: the staircase! Indeed, staircase form is a block-triangular form, but with non-square diagonal blocks.

Suppose, then, that the P3 heuristic is adapted to find a spike structure in the non-square diagonal blocks, B_{kk}, of the staircase, rather than in the square diagonal blocks of the block-triangular reduction. For a lower staircase, the new P3 would naturally be applied first to B_{11}, then B_{22}, and so forth in period order through B_{tt}.

Applied to B_{11}, which has full row rank, P3 must find a pivot element in every row. There may be some leftover columns, however, that cannot be assigned to pivot on any period-1 row.

Applied to B_{22}, P3 must find (rank B_{22}) pivots. If any period-2 rows then remain unassigned, they may be pivoted on some of the leftover columns from period 1. There may also be some leftover period-2 columns that cannot be assigned to pivot on any period-2 row.

The situation is similar as P3 is applied to any block B_{kk}. It must be possible to assign (rank B_{kk}) pivots. If any period-k rows then remain unassigned, they may be pivoted on some of the leftover columns from previous periods. There may also be some leftover period-k columns that will have to pivot on rows of later periods.

Provided that B is nonsingular, this procedure can always be carried out through B_{tt} to yield a complete assignment of pivots: nonsingularity guarantees that, for each k, the rows of periods 1 through k must be able to pivot on just the columns of periods 1 through k. The result is a bump-and-spike structure having two kinds of spike columns: <u>intraperiod spikes</u> of period k that are assigned by P3 to pivot on period-k rows; and <u>interperiod spikes</u> of period k, the leftovers that must pivot on rows in later periods. Fill-in is confined to the spikes, as with standard bump-and-spike techniques.

Furthermore, it should be clear that the above procedure produces a natural pivot order in the sense of Proposition 7. Thus it appears that <u>both</u> disadvantages of standard bump-and-spike can be eliminated: P3 need only be applied to diagonal blocks of the staircase--whose size is manageable--and the resulting pivot order is a natural one for staircase structures.

The remainder of this section shows how the above outline can be developed into a practical technique. Two adaptations of P3 to non-square matrices are presented: one applies the heuristic to the entire matrix, while the other decomposes the matrix and applies P3 to blocks within it. Other matters that must be considered are choice of pivot rows for interperiod spikes, and handling of pivots that are zero or too small during elimination. All of these topics are considered in the context of first-order staircase systems, but the same procedures apply virtually unchanged to higher-order systems.

Computational experiments, reported at the end of this section, compare a standard bump-and-spike algorithm with one version of a staircase adaptation.

Adaptation of P3

Hellerman and Rarick's original algorithm P3 [27] is a heuristic for rearranging a nonsingular matrix to have a good bump-and-spike form. Applied to the diagonal blocks of a block-triangular reduction, P3 necessarily finds just one bump per block. In general, howevei, P3 tries to find numerous small bumps, as many of size 1 as possible. Hence it is appropriate to try to apply P3 to the diagonal blocks of a staircase --provided that the algorithm is adapted to handle matrices that are not necessarily square or of full rank.

P3 is a preselection algorithm. Given a square matrix A, it chooses an ordering of the rows and columns based only on information at hand before any pivots are carried out. In particular it needs to know the position of each nonzero element in A, the number of nonzero elements in each column (the "column count"), and the number of nonzero elements in each row (the "row count"). Rows and columns are chosen for ordering one at a time. As each is chosen, the relevant counts are decremented accordingly; thus, at any point in the algorithm, the counts refer only to nonzeroes in the <u>remaining</u> unchosen rows and columns.

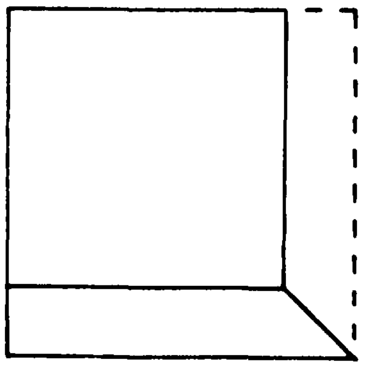
The underlying strategy of P3 is to create triangle columns whenever possible; spikes are created only when there is no other choice. Accordingly, the standard P3 has three main stages:

(1) Choose columns that have a count of 1, and assign them and their pivot rows toward the end of A.

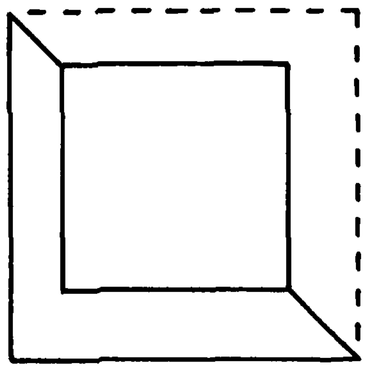
(2) Choose rows that have a count of 1, and assign them and their pivot columns toward the beginning of A.

(3) Choose a spike column; it will ultimately be the last column of a bump toward the beginning of A. Choose other columns and rows to fill out the bump, trying to keep it small and sparse.

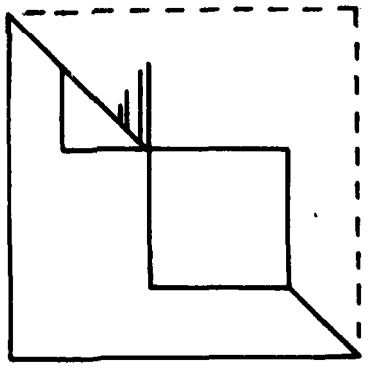
Stage 1 is repeated as many times as possible, to produce a set of triangle columns at the end of A:



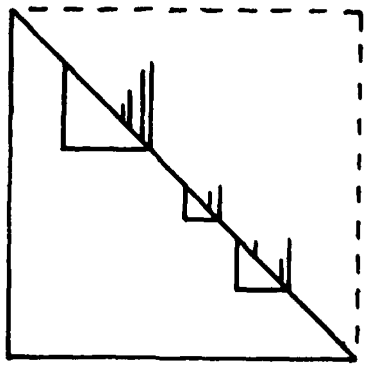
Stage 2 is then repeated in a similar manner to produce a set of triangle columns at the beginning of A:



Finally, stage 3 adds a bump after the initial triangle columns:



Control now returns to stage 2, which adds more triangle columns after the bump; then stage 3 produces another bump; and stages 2 and 3 continue to alternate until no rows and columns remain:



(Stages 2 and 3 operate in such a way that there is never occasion to return to stage 1.)

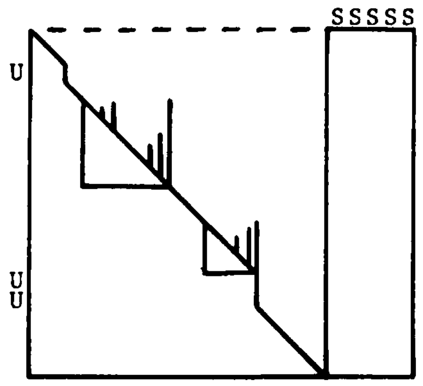
Adapting P3 to handle staircase blocks requires only one significant change to this outline: stage 2 must accommodate rows whose counts fall to zero. The adapted stage 2 has two parts:

(2a) Choose rows that have a count of 0, and assign them toward the beginning of A. Mark them as unpivoted.

(2b) Choose a row that has a count of 1, and assign it and its pivot column toward the beginning of A.

One pass through stage 2 now proceeds as follows: repeat 2a until there are no zero-count rows, then carry out 2b for any 1-count row. As before, the entire stage 2 is repeated as many times as possible, and alternates with stage 3.

The adapted P3 stops when all rows have been chosen. It puts A into a bump-and-spike form like the following:



U: unassigned rows

S: unassigned columns (spikes)

In general, most rows and columns are paired off to pivot within A. However, some columns may remain unchosen, and some rows may be marked unpivoted.

It is now easy to see how the adapted P3 finds a natural staircase pivot order. Applied to block B_{kk}, it pivots most rows of the block on columns of the block; these are the triangle columns and intraperiod spikes of period k. If any unchosen columns remain, they become the interperiod spikes from period k. If there are any unpivoted rows, they are assigned pivots on interperiod spikes from previous periods.

When P3 is applied in this way to blocks B_{11} through B_{tt}, in order, the result is for all practical purposes a natural pivot order It should be noted, however, that the number of pivots chosen by P3 within B_{kk} may be greater than the rank of B_{kk}, since P3 looks only at the pattern of nonzeroes within B_{kk} and not at their values. If this is the case, however, certain intraperiod spikes of period k will turn out to have pivots of zero during the elimination; the situation will be remedied by swapping these spikes with later interperiod spikes, as described later on in this section. The actual pivot order, after spike swapping, is indeed a natural pivot order in the sense of Proposition 7.

P3 algorithm for staircase blocks

It remains to state more fully the adaptation of P3 outlined above. For this purpose, R_i and C_j will be the row and column counts in the remaining unchosen columns of $m \times n$ matrix A. The first and last remaining rows will be μ_0 and μ, and the first and last remaining columns will be ν_0 and ν.

The main routine for P3 is then as follows:

```
P3:  1: SET $R_i$ = number of nonzeroes in row i of A
        SET $C_j$ = number of nonzeroes in column j of A
     2: REPEAT WHILE $C_j = 1$ for any remaining column j:
        Find the i for which $A_{ij} \neq 0$
        Choose j as the $\nu$th column; set $\nu = \nu - 1$
        Choose i as the $\mu$th row; set $\mu = \mu - 1$
        Set $C_k = C_k - 1$ for each remaining column k such that $A_{ik} \neq 0$
     3: REPEAT WHILE $\mu_0 \leq \mu$:
        3.1: REEPAT WHILE $R_i \leq 1$ for any remaining row i:
             3.1.1: REPEAT WHILE $R_i = 0$ for any remaining row i:
                    Choose i as the $\mu_0$th row; set $\mu_0 = \mu_0 + 1$
                    Mark $\mu_0$ to be an unpivoted row
        3.1.2: IF $R_i = 1$ for any remaining row i:
               Find the j for which $A_{ij} \neq 0$
               Choose i as the $\mu_0$th row; set $\mu_0 = \mu_0 + 1$
               Choose j as the $\nu_0$th column; set $\nu_0 = \nu_0 + 1$
               Set $R_k = R_k - 1$ for each remaining row k
                   such that $A_{kj} \neq 0$
    3.2: CALL SPIKE

  4: IF $\nu_0 \leq \nu$, mark remaining columns to be interperiod spikes.
```

SPIKE is a recursive routine that is called whenever all remaining rows and columns have counts of at least 2. It can be adapted to non-square A with only minor changes:

```
SPIKE: S1: CALL CHUZS to choose a remaining column s to be a spike
           Set $R_k = R_k - 1$ for each remaining row k such that $A_{ks} \neq 0$
       S2: REPEAT WHILE $\mu_0 \leq \mu$, $\nu_0 < \nu$:
           S2.1: IF $R_i > 1$ for every remining row i: CALL SPIKE
           S2.2: If $R_i = 1$ for some remaining row i:
                 CALL CHUZJ to pick a column j for which $A_{ij} \neq 0$
                     and $R_i = 1$
                 Choose i as the $\mu_0$th row; set $\mu_0 = \mu_0 + 1$
                 Choose j as the $\nu_0$th column; set $\nu_0 = \nu_0 + 1$
                 Set $R_k = R_k - 1$ for each remaining row k such
                     that $A_{kj} \neq 0$
```

```
S2.3: If $R_i = 0$ for some remaining row i:
      Choose i as the $\mu_0$th row; set $\mu_0 = \mu_0 + 1$
      Make s the $\nu_0$th column; set $\nu_0 = \nu_0 + 1$
      RETURN
```

For best results, CHUZS should pick a column s that intersects many low-count rows; then there is a good chance that one of these rows will soon be reduced to a count of zero, whereupon SPIKE will reach step 2.3 and return. The same applies to CHUZJ; in particular, step 2.3 will be reached immediately if the chosen column j intersects two or more 1-count rows. Consequently, both CHUZS and CHUZJ may be as follows:

```
CHUZS, CHUZJ:
    C1: FOR each remaining column k:
          FOR $\ell$ FROM 1 TO $(\nu-\nu_0)$:
            SET $T_\ell(k)$ = number of remaining rows i such that
                    $A_{ik} \neq 0$ and $R_i = \ell$
    C2: RETURN a column k for which the vector
        $(T_1(k), T_2(k), \ldots, T_{\nu-\nu_0}(k))$ is lexicographically
        maximal among all remaining columns k
```

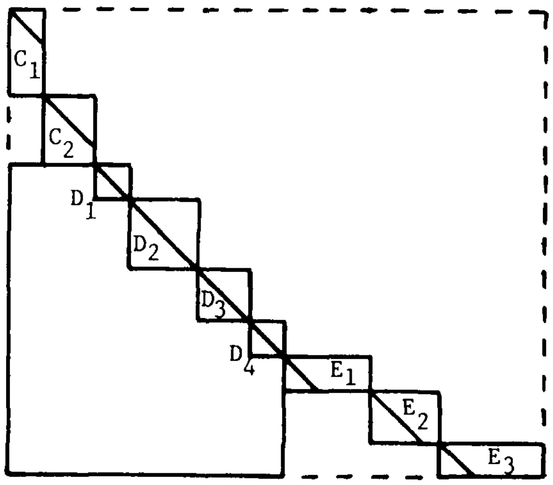
(In their original proposal [27] Hellerman and Rarick employ a similar, but not identical, criterion; they also indicate how the recursion of SPIKE may be implemented efficiently.)

The output of P3 is a reordering of A. The chosen pivots are the elements (μ,ν) or (μ_0,ν_0) defined in steps 2, 3.1.2, S2.2 and S2.3; these are exactly the elements that lie on the diagonal of the reordered A if unpivoted rows and interperiod spikes are disregarded.

Canonical reduction of staircase blocks

Provided that the staircase diagonal blocks are of moderate size--say, at most 100 or 150 rows--applying P3 to each block should be adequately efficient. When the diagonal blocks are very large, however, it may be worth taking the following more elaborate approach that applies P3 only to sub-blocks.

Just as a nonsingular square matrix has a square-block-triangular reduction, a general $m \times n$ matrix has a canonical reduction with non-square blocks along the diagonal. (Dulmage and Mendelsohn showed the essential uniqueness of this reduction in their study of bipartite graphs [18,19].) A canonically reduced matrix has a block structure like this:



Each block C_ℓ has more rows than columns, and possesses a maximum transversal (that is, there is a permutation of the rows of C_ℓ such that all diagonal elements are nonzero). Each block D_ℓ is square and has a maximum transversal. Each E_ℓ has more columns than rows and possesses a maximum transversal (there is a permutation of the columns of E_ℓ such that all diagonal elements are nonzero).

Since in practice a staircase block B_{kk} tends to have a low proportion of nonzeroes, it is reasonable to expect that the canonical reduction of B_{kk} will have numerous small diagonal blocks C_ℓ, D_ℓ, and E_ℓ. If some way can be found to apply P3 to each of these blocks--instead of to B_{kk} as a whole--considerable savings may be realized when B_{kk} is large.

Consider, then, first applying P3 to one of the blocks C_ℓ. Since C_ℓ has a maximum transversal, any p columns of C_ℓ must intersect at least p different rows. As a consequence, P3 cannot produce any interperiod spikes from C_ℓ: it could do so only by reducing a column count to zero in step 2 or by running out of rows in SPIKE; but either of these cases yields a set of p columns that intersects fewer than p rows. Thus P3 assigns a pivot on every column of C_ℓ, and necessarily leaves some unpivoted rows to be assigned pivots on interperiod spikes from previous periods.

The square blocks D_ℓ have the same properties as the diagonal blocks of a block-triangular reduction. Hence P3 may be applied to these blocks in the usual way, and will find a pivot on every row and column.

Finally, consider applying P3 to one of the blocks E_ℓ. Because E_ℓ has a maximum transversal, any p rows of E_ℓ must intersect at least p different columns. So, in this case, P3 cannot leave any unpivoted rows: it could do so only by reducing a row count to zero in step 3.1.1, which would identify a set of p rows that intersect fewer than p columns. Thus P3 assigns a pivot on every row of E_ℓ, and leaves some columns to be interperiod spikes.

These observations suggest a four-step procedure for pivot assignment in staircase block B_{kk}:

(1) Find the canonical reduction of B_{kk}.

(2) Apply P3 to choose a pivot in every column of each block C_ℓ. Assign interperiod spikes from previous periods to pivot on the leftover rows.

(3) Apply P3 in sequence to D_1, D_2, ... to choose a pivot in every row and column of these blocks.

(4) Apply P3 to choose a pivot in every row of each block E_ℓ. Save the leftover columns to be interperiod spikes from period k.

This scheme tends to pivot old interperiod spikes in earlier rows of B_{kk}, while it creates new interperiod spikes from the later columns. Thus it promotes sparsity as well as efficiency. It should be noted, however, than any of the steps 2, 3, and 4 above may be empty. In particular, if B_{kk} has full row rank then the canonical reduction has no blocks C_ℓ, and if B_{kk} has full column rank then there are no E_ℓ.

Success of the above procedure certainly depends upon an efficient algorithm for step 1: the work in finding a canonical reduction must not exceed the savings in applying P3 to only the small sub-blocks. Fortunately, as is clear from [19], finding the canonical reduction is a straightforward extension of finding a square-block-triangular reduction. Indeed the two-part algorithm for the latter, set forth in Section 1, is readily extended and adapted to create a three-part algorithm for canonical reduction of an $m \times n$ matrix A:

(1) Find the longest possible transversal in A. The transversal algorithm of [12] can be adapted to this purpose without difficulty.

(2) Identify the blocks C_ℓ and E_ℓ. This is a simple and straightforward procedure described in [19].

(3) Find the blocks D_ℓ, by applying a strong-components algorithm [16,26] to the submatrix of rows and columns that are not in any C_ℓ or E_ℓ.

If experience with nonsingular matrices can be a guide, the time required by these steps should not dominate the time required by P3.

<u>Pivot choice for interperiod spikes</u>

P3 generally leaves $(m_k - \text{rank } B_{kk})$ rows of $B_{\cdot k}$ unpivoted, while there are $\sum_{i=1}^{k-1} (n_i - m_i)$ interperiod spikes available to pivot on these rows. Hence often there is a choice to be made: which interperiod spike (of periods $1, \ldots, k-1$) should pivot on each leftover row (of period k)?

If the staircase is fully dense, the answer is clear from Proposition 6: it just does not matter which interperiod spike is assigned to which row. The total fill-in is the same regardless.

For the sparse staircases that arise in practice, nothing so definite can be said. Proposition 7 gives an upper limit on fill-in that is independent of where the interperiod spikes are pivoted; but fill-in could vary considerably within the limit. Still, it is reasonable to expect that, over a variety of practical problems, any particular way of pivoting interperiod spikes will not be uniformly much better than any other way. A few informal tests have tended to bear this out.

Thus any simple rule for pivoting interperiod spikes should be acceptable. One easily implemented rule, for example, is to always choose the shortest spike available. Unpivoted interperiod spikes are stored in a "stack", in order of height, with the shortest at the top. If there are unpivoted rows in B_{kk}, a spike is unstacked to pivot on the first one, then another is unstacked to pivot on the second, and so forth. Then, if there are new interperiod spikes from period k, they are added to the stack, shortest last. This is the method used in the computational experiments reported below.

Spike swaps

Once an initial pivot order has been arrived at, it remains to actually carry out the elimination. In the process some pivot elements may be found unacceptable, and if so the order must be revised. The row order, however, must be kept to preserve the fill-in properties of Proposition 7 and the bump-and-spike structure. Thus any adjustments must be made, as for standard bump-and-spike methods, by exchanging (or "swapping") columns.

As explained in Section 1, there are two tests of each potential pivot element $\beta_{kk}^{(k)}$. First the pivot's magnitude must be greater than some absolute tolerance ϵ; second, the pivot's ratio to $\max_{\ell} |\beta_{\ell k}^{(k)}|$--the largest remaining element in its column--must not exceed in magnitude some (usually larger) relative tolerance δ.

If k is a triangle column it cannot fill in, and so $\beta_{\ell k}^{(k)} = [B]_{\ell k}$ for all rows $\ell \geq k$. Thus if B is adequately scaled all pivots in triangle columns will be acceptable, and rejected pivots will be found only in the spike columns. (This was in fact the case for all test problems described below except GROW15 and GROW22, for which some "triangle swaps" were necessary to preserve stability.)

A spike column's pivot can fail the relative tolerance test only because the pivot row is poorly chosen from a numerical standpoint. Failing the absolute test may also be a sign of numerical problems; however, if B is at all well-conditioned, a spike is much more likely to fail the absolute test because its pivot is actually zero. There are several ways that a spike column k may have a zero pivot:

- P3 may assign a pivot element that turns out to be zero: sometimes $[B]_{kk} = 0$ and fails to fill in; sometimes $[B]_{kk} \neq 0$ but is cancelled to zero.
- An interperiod spike may be assigned a pivot element that turns out to be zero. Indeed, if a spike from period ℓ pivots in period $\ell+2$ or later, necessarily $[B]_{kk} = 0$, and there will be a zero pivot unless it fills in.
- The number of pivots chosen in block $B_{\ell\ell}$ may be greater than the rank of $B_{\ell\ell}$. Some of the pivots must then turn out to be zeroes.

A small number of rejected spikes is thus usually unavoidable, no matter how well-behaved B is numerically.

When the pivot in spike k is rejected, column k must be swapped with a later, more acceptable column j; $\beta_{kj}^{(k)}$ then becomes the pivot element. $\beta_{kj}^{(k)}$ must be zero, however, if j is a triangle column or a spike from a later period, and hence k can only be swapped with a spike in the same or an earlier period. It follows that, if k is an interperiod spike, it must be swapped with another interperiod spike. If k is an intraperiod spike, however, there are several possiblities: it may be swapped with another intraperiod spike; it may be swapped with an interperiod spike j of the same period, in which case k becomes an interperiod spike and j becomes an intraperiod spike; or it may be swapped with an interperiod spike j of a previous period, in which case k becomes an interperiod spike and j remains one. Proposition 7 suggests that none of these swaps should significantly increase fill-in in the majority of cases, since the bound on fill-in is independent of the number and position of interperiod spikes.

Usually there will be several spikes eligible for swapping with k, and a good choice will have to be made among them. To promote stability, the new pivot's ratio to $\max_{\ell}|\beta_{k\ell}^{(k)}|$--the largest potential pivot in the same row--should not exceed in magnitude the tolerance δ. Otherwise, choice of a swap spike can be made heuristically; any strategy appropriate to the standard bump-and-spike method will also be appropriate to the staircase methods presented here. (Tests described below adopt the spike-swapping strategy of MINOS [47].)

Computational experience: sparsity

Test groups 1 and 3 (as described in Section 3) provided evidence upon the relative sparsity, stability and speed of the standard and staircase bump-and-spike elimination methods. Experiments with the staircase method employed the simpler version in which P3 is applied to each entire diagonal block B_{kk}; other details are given in Appendix B.

Overall the results were about as expected. The staircase approach was superior for large-bump systems having a good staircase structure. For small-bump systems the standard approach came out ahead.

The first table below shows the numbers of nonzeroes in the L and U factors, and the fill-in (the percentage increase in number of nonzeroes in L and U over number of nonzeroes in B). All figures are averages from test group 1, and may be compared with the statistics in Appendix C:

	L		U		% FILL-IN	
	STD	STAIR	STD	STAIR	STD	STAIR
GROW15	1965	1677	1202	1268	42%	32%
GROW22	3742	3189	2152	2452	36%	31%
SCSD8-	1149	1126	237	410	26%	40%
SCSD8+	1196	1170	353	550	36%	51%
SCSD8++	1194	1167	442	688	45%	64%
SCAGR25-	1401	1355	54	268	6%	18%
SCAGR25+	1379	1360	121	255	11%	20%
SCRS8-	1324	1314	125	385	17%	37%
SCRS8+	1433	1292	306	463	42%	44%
SCFXM2-	2497	2475	194	551	12%	26%
SCFXM2+	2676	2602	314	860	19%	38%
PILOT-	5086	3825	2747	3011	163%	129%
PILOT+	6044	3844	3465	3274	219%	139%
BP1-	8352	7293	2360	4576	142%	168%
BP1+	9494	7562	3037	5104	181%	184%

There is an evident tradeoff between sparsity in L and in U. The staircase method tends to give a sparser L, consistent with Proposition 7's restrictions on fill-in, but a denser U, owing probably to fill-in within large interperiod spikes. Hence either method can be sparser under the proper conditions.

The staircase method seems clearly sparser for the PILOT systems --which had the largest bumps--and for systems having the GROW15-GROW22 structure. For the large-bump systems of BP1 and SCRS8+ the two methods are roughly tied; the staircase method would almost surely improve on BP1, however, if a better staircase partition were determined for it. The SCSD8 systems were the only fairly large-bump ones for which the standard method is definitely sparser; it is also sparser in all of the smaller-bump cases. In no event, however, does the staircase method produce a disastrously large *fill-in*.

Computational experience: stability

To gauge stability, the relative tolerance test was monitored and the smallest pivot ratio greater than δ was recorded. In addition, after each elimination the solution-check routine of MINOS was invoked. This routine uses the LU factorization to solve $Bx = b$, with b the LP's right-hand side, and reports the largest component of the error $e = b - Bx$; it also performs one step of iterative refinement, solving $By = e$ and correcting x to $x + y$, and from the relative sizes of x and y it determines a lower bound on the condition number of B.

In the test runs the standard method has a generally higher minimum pivot ratio for four of the LPs--SCAGR25, SCRS8, SCSD8, and

SCFXM2--but only for SCFXM2 did it have a smaller solution-check error as well. The standard method also gave a smaller error for GROW15-GROW22 and PILOT; these LPs had minimum pivot ratios near δ for both methods.

On the whole the bases of SCAGR25, SCSD8 and SCRS8 were well-conditioned, SCFXM2 and BP1 were fairly conditioned, and GROW15-GROW22 and PILOT were poorly conditioned. There was considerable variation for some LPs, however, notably SCAGR25. Four bases had to be refactorized with higher δ when their initial factorizations failed a solution check: both methods failed for a SCAGR25 basis, the standard method failed on one SCRS8 basis and one SCFXM2 basis, and the staircase method failed on a PILOT basis.

A related statistic is the number of spikes rejected due to unsatisfactory pivots:

	REJECTED SPIKES	
	STD	STAIR
GROW15	13	0
GROW22	22	1
SCSD8-	2	3
SCSD8+	2	7
SCSD8++	2	7
SCAGR25-	0	1
SCAGR25+	3	6
SCRS8-	0	8
SCRS8+	1	5
SCFXM2-	0	4
SCFXM2+	2	5
PILOT-	17	10
PILOT+	22	11
BP1-	20	47
BP1+	18	45

The large-bump, poorly-conditioned problems--GROW15-GROW22 and PILOT--show clearly fewer rejected spikes in the staircase method; for BP1 the situation is reversed, probably because of its poor staircase structure. Small-bump problems show an edge for the standard method (as usual), but the difference is rather small.

In summary, it appears that either method is acceptably stable for linear programming, although the standard method sometimes looks better statistically. For other applications a higher δ would be advisable.

Computational experience: speed

Timings of the two methods show, as predicted, a wide gap in the efficiency of the P3 heuristic:

	MEDIAN COLS IN BIG BUMP	TIME IN P3: SEC/10 ELIM		% OF ELIM TIME IN P3		% OF TOTAL TIME IN P3	
		STD	STAIR	STD	STAIR	STD	STAIR
SCTAP2	1	.0	.7	0%	24%	0%	2%
SCRS8	28	.2	.2	19%	17%	1%	2%
SCFXM2	36	.2	.8	12%	27%	1%	4%
SCAGR25	45	.4	.2	27%	15%	3%	2%
SCSD8	114	1.1	.2	41%	12%	5%	1%
BP1	408	13.1	3.8	47%	10%	10%	3%
PILOT	533	20.4	2.4	62%	25%	16%	3%

As the size of the largest bump grows, the standard method's time in P3 shoots up dramatically; for the last three cases over 40% of the cost of the elimination is in P3, and for the worst case (PILOT) P3 was over 15% of the total cost of the simplex method! By contrast, the staircase approach keeps the cost of P3 fairly level, and its timings are particularly low when there are many small periods.

Significant times are also spent in actually computing the L and U factors. Here the cost is mainly a function of the numbers of spikes and rejects:

	AVG SPIKES		AVG SPIKES REJECTED		SEC/10 FACTORIZATIONS	
	STD	STAIR	STD	STAIR	STD	STAIR
SCTAP2	5	9	0	4	.3	1.2
SCRS8	13	16	0	5	.2	.7
SCFXM2	28	29	0	3	.5	1.3
SCAGR25	9	21	1	4	.3	.8
SCSD8	24	30	1	5	.6	.8
BP1	121	147	20	46	9.9	19.4
PILOT	93	97	21	11	9.5	5.8

For small-bump problems the standard method's advantage is large proportionately, small in absolute terms. For SCSD8 the difference in factorizing times is swamped by the difference in P3 times, and for PILOT the staircase method actually factorizes faster owing to fewer rejects. (BP1 again encounters difficulty with a poor staircase structure.)

Altogether Gaussian elimination involves P3, block-triangular reduction (for the standard method only), computation of L and U, and control and utility routines. Totals for all of these were as follows:

	SEC/10 ELIMINATIONS		
	STD	STAIR	% STAIR/STD
SCTAP2	1.7	3.0	+ 76%
SCRS8	1.1	1.4	+ 27%
SCFXM2	1.9	2.8	+ 47%
SCAGR25	1.4	1.6	+ 14%
SCSD8	2.7	1.6	- 31%
BP1	27.9	26.2	- 6%
PILOT	32.8	9.7	- 70%

The advantage switches clearly from the standard method, for small-bump problems, to the staircase method for large-bump ones. Even the BP1 bases are handled slightly faster.

These results establish that standard bump-and-spike elimination methods are unacceptably slow for large-bump staircase systems. For such systems a staircase method should produce an acceptable elimination in much less time.

6. A LOCAL-MINIMIZATION TECHNIQUE FOR STAIRCASE FLIMINATION

Motivation

Local-minimization techniques for pivot selection are readily adapted to find a natural staircase pivot order in the sense of Propositions 4 and 5. Moreover, the adapted techniques should require less time and storage than standard local-minimization techniques applied to staircase matrices.

A plan of adaptation can be derived straightforwardly from Proposition 4. The proof of this proposition shows that, to produce a natural pivot order, it suffices to just restrict the choice of pivot elements in certain ways. First, pivot elements are chosen only in the rows and non-linking columns of period 1; second, after the non-linking columns have been exhausted, pivots are chosen only in the rows and linking columns of period 1; third, after the rows have been exhausted, pivots are chosen only in the linking rows of period 2 and the linking columns of period 1. In this way all columns of period 1 are pivoted. The procedure then continues analogously for period 2: fourth, only the rows and non-linking columns of period 2; fifth, only the rows and linking columns of period 2; sixth, only the linking rows of period 3 and the linking columns of period 2. Periods 3 through t are treated in the same way.

The only flexibility in choosing a natural pivot order, therefore, lies in selecting pivot elements from the various restricted sets. This selection is easily made by any local-minimization technique: at each step, choose a pivot element that minimizes some merit function over all potential pivots in the current restricted set. In other words, local minimization is adapted to find a natural pivot order by simply restricting the minimization to certain subsets of potential pivots.

It follows immediately that the
more efficient than full local minimizati
minimizes the merit function over a much
to do.

It is also easy to see that the
have more modest storage requirements. S
keep track of all nonzeroes in the remair
since fill-in may occur anywhere within t
staircase order in the sense of Propositi
strictly limited so that most of $\beta^{(k)}$ i
when pivots are in the columns of period
been pivoted already--there can be no fil
after period $\ell+1$. Thus at most two peri
a time.

Such reduced storage requirement
For the standard approach (as explained i
L and U must generally share storage, wit
management routines and a fair-sized "cus
staircase adaptation, however, could very
part of $\beta^{(k)}$ in a separate region, savi
and storage-cushion space.

Algorithm

It is not hard to define an algo
tions. The algorithm's main steps are as

```
1:  SET $\ell = 1$
2:  REPEAT FOR  k  FROM 1 TO  m:
    2.1:  IF any non-linking column
          2.1.1:  Minimize  MERIT(i,
                  i  a period-$\ell$ row
                  linking column
          2.1.2:  Choose a pivot  (i
                  minimum
    2.2:  ELSE IF any row of period
          2.2.1:  Minimize  MERIT(i,
                  i  a period-$\ell$ row
          2.2.2:  Choose a pivot  (i
                  minimum
    2.3:  ELSE IF any column of peri
          2.3.1:  Minimize  MERIT(i,
                  i  a period-($\ell$+1)
                  period-$\ell$ column
          2.3.2:  Choose a pivot  (i
                  minimum
    2.4:  ELSE SET $\ell = \ell+1$.
```

Any of the merit functions described in Secti
for the function MERIT.

If the storage economies of this alg
it is worth working out exactly what part of
at each step. In general, it is only necessa

and columns of $\beta^{(k)}$ that might possibly have filled in. These are easily seen to depend only on the current value of ℓ, and on which step is being executed:

2.1: Only period-ℓ rows and columns can fill in.

2.2: Only period-ℓ rows, period-(ℓ+1) linking rows, and period-ℓ linking columns can fill in.

2.3: Only period-(ℓ+1) linking rows, period-ℓ linking columns and period-(ℓ+1) columns can fill in.

Thus the minimum maintained part of $\beta^{(k)}$ is expanded in three small steps as the columns of each period are pivoted.

Simplified algorithm for full staircase

If non-linking rows and columns are not distinguished from linking ones, step 2.1 of the preceding algorithm may be dropped and the remainder may be simplified:

```
F1:  SET $\ell = 1$
F2:  REPEAT FOR  k  FROM 1 TO  m:
     F2.1:  IF any row of period $\ell$ remains unpivoted:
            F2.1.1:  Minimize  MERIT(i,j)  subject to  $\beta^{(k)}_{ij} \neq 0$,
                     i  a period-$\ell$ row and  j  a period-$\ell$ column
            F2.1.2:  Choose a pivot  $(i_k,j_k)$  that achieves the
                     minimum
     F2.2:  ELSE IF any column of period $\ell$ remains unpivoted:
            F2.2.1:  Minimize  MERIT(i,j)  subject to  $\beta^{(k)}_{ij} \neq 0$,
                     i  a period-($\ell$+1) row and  j  a period-$\ell$
                     column
```

F2.2.2: Choose a pivot (i_k, j_k) that achieves the minimum

F2.3: ELSE SET $\ell = \ell + 1$

This algorithm just "pivots down the staircase" one block at a time: pivot elements are first restricted to B_{11}, then B_{21}, then B_{22}, B_{32}, B_{33}, ... and so on through $B_{t-1,t-1}$, $B_{t,t-1}$, and B_{tt}.

It is also easy to describe what part of $\beta^{(k)}$ can fill in. If the algorithm is in step F2.1, fill-in is confined to period-ℓ and period-$(\ell+1)$ rows and period-ℓ columns. In step F2.2 fill-in is confined to period-$(\ell+1)$ rows, and period-ℓ and period-$(\ell+1)$ columns.

Numerical considerations

To make the above algorithms numerically stable, the minimization must be confined to acceptable pivots in an absolute and relative sense as defined in Section 1. Thus the condition $\beta_{ij}^{(k)} \neq 0$ must be replaced by $|\beta_{ij}^{(k)}| \geq \varepsilon$; and it is necessary to add either a row ratio test, $|\beta_{ij}^{(k)}| \geq \delta \cdot \max_\ell |\beta_{i\ell}^{(k)}|$, or a column ratio test, $|\beta_{ij}^{(k)}| \geq \delta \cdot \max_\ell |\beta_{\ell j}^{(k)}|$.

Proposition 4 guarantees that an acceptable pivot order can always be found (if ε is not set too large), provided that the appropriate ratio test is conducted at each step. For example, in step 2.1.1 the minimization is over all nonzero elements of $\beta^{(k)}$ in the period-ℓ non-linking columns, but over only some nonzeroes of $\beta^{(k)}$ in the period-ℓ rows. Thus there must be pivots that satisfy the column-ratio test, but it is possible (though not likely) that all eligible pivots fail the row-ratio test. It follows that the column-ratio test should be used in 2.1.1; similar reasoning shows that 2.3.1 should also use the column-ratio test, while 2.2.1 should use the row-ratio test. Stable versions of these steps are thus as follows:

2.1.1: Minimize MERIT(i,j) subject to
$|\beta_{ij}^{(k)}| \geq \varepsilon$, $|\beta_{ij}^{(k)}| \geq \delta \cdot \max_{\ell} |\beta_{\ell j}^{(k)}|$,
i a period-ℓ row and j a period-ℓ nonlinking column

2.2.1: Minimize MERIT(i,j) subject to
$|\beta_{ij}^{(k)}| \geq \varepsilon$, $|\beta_{ij}^{(k)}| \geq \delta \cdot \max_{\ell} |\beta_{i\ell}^{(k)}|$,
i a period-ℓ row and j a period-ℓ column

2.3.1: Minimize MERIT(i,j) subject to
$|\beta_{ij}^{(k)}| \geq \varepsilon$, $|\beta_{ij}^{(k)}| \geq \delta \cdot \max_{\ell} |\beta_{\ell j}^{(k)}|$,
i a period-$(\ell+1)$ linking row and j a period-ℓ column

A similar analysis applies to the simplified full-staircase algorithm. The row-ratio test should be used in step F2.1.1, and the column-ratio test in F2.2.1.

In practice the arrangement of storage may make it inconvenient to do both row and column ratio tests (see, for example, [17]). Instead one of the tests may be used exclusively, with provisions for some sort of recovery in the rare event that all eligible pivots fail the test. The computations reported below took this approach--only column-ratio tests were applied--and encountered no difficulty: there was not a single recorded case of all eligible pivots failing the test.

Higher-order staircase systems

In general the algorithms of this section break down when there are nonzero elements below the staircase: it can easily happen that all remaining elements of $\beta^{(k)}$ in the current restricted subset are zero. In such an event a pivot must be sought elsewhere, preferably in a way that permits a return to the natural pivot order as soon as possible.

Even if the algorithms are modified to find pivots in every case, the fill-in associated with a higher-order staircase will be more extensive. Consequently the advantages in storage and speed for a first-order system will tend to be lost with higher-order ones.

In light of these observations it seems that the adapted local-minimization methods will mainly be useful for first-order staircase systems, while high-order systems are more amenable to a bump-and-spike approach.

Computational experience

Test group 2 provided some initial evidence for the staircase adaptations. Statistics were collected for both the reduced and full algorithms, as well as for standard local minimization. Tables below may be compared with additional data in Appendix D.

Average fill-in was as follows (STD indicates standard, F.ST. full staircase, and R.ST. reduced staircase):

	L			U			% FILL-IN		
	STD	F.ST.	R.ST.	STD	F.ST.	R.ST.	STD	F.ST.	R.ST.
GROW15	1398	1321	1392	1703	1775	1413	39%	39%	26%
SCSD8-	762	888	887	529	612	626	26%	46%	47%
SCSD8+	858	1123	1137	674	719	731	36%	63%	65%
SCAGR25-	916	934	909	469	443	471	6%	5%	6%
SCAGR25+	776	892	889	684	616	614	8%	11%	11%
SCRS8-	852	1193	1166	651	619	646	22%	47%	47%
SCRS8+	843	1139	1116	643	632	673	22%	46%	46%
SCFXM2-	1440	1506	1438	952	960	1038	9%	12%	13%
SCFXM2+	1488	1570	1500	1285	1320	1440	12%	17%	19%

The staircase methods yielded significantly more fill-in for SCSD8 and SCRS8, and only slight more for SCAGR25 and SCFXM2; for GROW15 the fill-in was significantly less for the reduced staircase method. Notably, GROW15 had the densest diagonal blocks B_{kk}, but few nonzeros in off-diagonal blocks $B_{k+1,k}$; SCAGR25 and SCFXM2 also had many more nonzeroes in diagonal than off-diagonal blocks. SCSD8 and SCRS8 had the highest proportions of off-diagonal nonzeroes, which may have contributed to propagation of fill-in down the staircase.

The full and reduced staircase methods gave about the same results except for GROW15: the special structure of this LP (Appendix A) places most of the density in the non-linking columns, so that the reduced-staircase algorithm yields a quite different pivot order. The relative density of L and U fluctuates between methods with no apparent pattern.

All eliminations were subjected the stability tests described in Section 5. Solution check errors were uniformly satisfactory, and no factorizations were rejected; the errors were sometimes smaller for the staircase methods with SCFXM2, and were definitely smaller with the reduced staircase method for GROW15. Minimum pivot ratios were all fairly small.

To make these runs, local-minimization routines were added to the MINOS LP system in a crude but simple way, as a sophisticated implementation would have required much more extensive modifications. Consequently, there was no point in making timings. It does seem fairly certain, however, that the staircase methods would enjoy some advantage in speed and storage; the size of this advantage remains to be determined.

7. CONCLUSIONS

Comparison of elimination techniques

To compare structural and local-minimization techniques, the bases of test group 2 were factored both ways. Resulting fill-in was as follows:

	BUMP & SPIKE (% FILL-IN)		LOCAL MINIMIATION (% FILL-IN)		
	STD	STAIR	STD	F.ST.	R.ST.
GROW15	42%	32%	39%	39%	26%
SCSD8-	25%	48%	26%	46%	47%
SCSD8+	39%	53%	36%	63%	65%
SCAGR25-	6%	17%	6%	5%	6%
SCAGR25+	13%	23%	8%	11%	11%
SCRS8-	18%	39%	22%	47%	47%
SCRS8+	41%	44%	22%	46%	46%
SCFXM2-	14%	27%	9%	12%	13%
SCFXM2+	21%	43%	12%	17%	19%

The general impression is that, for these test problems neither technique has an overwhelming edge; a decision between them might well be made on other grounds such as speed or convenience.

For three of the problems there is some indication, however, that local minimization has a greater advantage on large-bump systems (SCAGR25+, SCRS8+, SCFXM2+) than on small-bump ones (SCAGR25-, SCRS8-, SCFXM2-). This stands to reason: success of the structural approach is predicated upon a structure of small bumps, and it should get progressively worse as bump size increases. Local minimization ignores the bump structure and is relatively unaffected by it.

Indications of numerical stability were also about the same for the two techniques. Local minimization tended to produce somewhat smaller pivot ratios, but solution-check errors were generally about the same. The one exception was GROW15, for which local-minimization gave much smaller errors.

It has been observed that stability tends to trade off against fill-in according to the setting of the relative tolerance δ [17,54]: a small δ (as in these experiments) tends to favor low fill-in, while a large δ favors stability. Hence the above comparisons are only suggestive; results for a range of δ values will be required for more definitive conclusions.

As for the relative speed of these two techniques, virtually nothing is known: they are almost never implemented together, and no comparable timings have been reported. A careful parallel implementation, perhaps within a linear-programming code, will be needed to properly contrast the efficiency of structural and local-minimization approaches.

Outlook for staircase techniques

The theoretical and experimental results of this paper suggest that staircase elimination approaches have three potential advantages over standard sparse-elimination techniques.

First, staircase methods can produce sparser L and U factors for certain kinds of staircase systems. When bump-and-spike structural techniques are employed, the staircase adaptation appears sparser on "large-bump" systems--ones whose block-triangular reductions have a few large diagonal blocks. The staircase variants of local minimization may be sparser when the staircase is fairly dense, and especially when the density is concentrated is the diagonal blocks of the staircase. (In other situations these approaches may produce a somewhat denser L and U than standard methods, but the differences do not seem to be disastrously large.)

Second, staircase methods can be more economical. For large-bump systems, the standard bump-and-spike technique spends so much time analyzing the bump that it becomes prohibitively expensive compared to the staircase version. (For small-bump systems the standard technique is somewhat faster, but both methods are fast.) If local-minimization techniques are used, the staircase variants promise to require less storage and storage-management time for all but the sparsest systems.

Third, staircase methods find natural pivot orders--in which most or all of each period is pivoted before the following ones--and such orders may be taken advantage of to solve certain linear systems faster. This idea has been applied quite successfully to linear programming [22].

In sum, sparse Gaussian elimination of staircase structures is worth a special look. It appears that many staircase systems could be solved advantageously by special methods that take the staircase structure into account.

APPENDIX A: TEST PROBLEMS

The linear programs used for the computational experiments of Sections 3, 5 and 6 are described in greater detail below. The tabular summaries for each LP are largely self-explanatory, but a few general notes are appropriate:

All statistics except OBJ ELEMS refer only to the staircase constraint matrix, excluding the objective row and right-hand side. In each case the constraint matrix, A, has been put in reduced standard form; DIAGONAL BLOCKS refers to the staircase blocks $A_{\ell\ell}$, OFF-DIAGONAL BLOCKS to the blocks $\hat{A}_{\ell+1,\ell}$, and SUB-STAIR BLOCKS (when present) to the blocks $A_{\ell+2,\ell}, \ldots, A_{t\ell}$.

Variables (columns) are implicitly constrained only to be nonnegative, unless there is an indication to the contrary. BOUNDED implies implicit lower and upper bounds, FIXED implies fixture at a set value, and FREE implies no implicit constraints.

MAX ELEM and MIN ELEM are the largest and smallest magnitudes of elements in A; LARGEST COL RATIO is the greatest ratio of magnitudes of elements in the same column of A. Where values are given BEFORE SCALING and AFTER SCALING, all tests were conducted with A scaled as described in Appendix B. Otherwise NO SCALING is indicated.

GROW15

A simple dynamic input-output LP constructed for test purposes. It is based on the following model: define sets,

IND	set of goods
OBJ	set of export goods: a subset of IND

and parameters

T	number of periods
a_{ij}	units of good i needed to produce 1 unit of good j; i, j $\in$ IND
m_i	maximum production of good i in a period; i $\in$ IND
σ	proportion of m_i that may be stored: σm_i is maximum stock of good i at beginning of a period
p_{it}	expected profit per unit of good i in period t; i $\in$ OBJ, t = 1,..., T.

Then the variables are

x_{it}	production of good i in period t; i $\in$ IND, t = 1,..., T
s_{it}	stock of good i at the beginning of period t; i $\in$ IND, t = 1,..., T+1

and the LP is

$$\text{maximize} \quad \sum_{t=1}^{T} \sum_{i\in OBJ} p_{it}x_{it}$$

$$\text{subject to} \quad s_{i,t+1} = s_{it} + x_{it} + \sum_{j\in IND} a_{ij}x_{jt}, \quad i\in \text{IND}, t=1,\dots,T$$

$$0 \le x_{it} \le m_i \qquad j\in \text{IND}, t=1,\dots,T$$

$$0 \le s_{it} \le \sigma m_i \qquad i\in \text{IND}, t=1,\dots,T+1$$

For GROW15, T = 15. The values of a_{ij} and m_i were taken from a 20-sector input-output analysis of the U. S. economy in [25]. A set OBJ of size 3 was picked arbitrarily, as were the values p_{it}; σ was set at 0.3.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1-14	20	43	356	41%	20	20	20	5%	3
15	20	43	356	41%					3
			5340	41%			280	5%	45

GRAND TOTALS		
ROWS	300	(ALL EQUALITIES)
COLS	645	(510 BOUNDED)
ELEMS	5620	
DENS	2.9%	

COEFFICIENTS	NO SCALING
MAX ELEM	1.0
MIN ELEM	6.0×10^{-6}
LARGEST COL RATIO	1.3×10^{5}

AD-A081 856 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 12/1
SPARSE GAUSSIAN ELIMINATION OF STAIRCASE LINEAR SYSTEMS.(U)
NOV 79 R FOURER N00014-75-C-0267
UNCLASSIFIED SOL-79-17 NL

2 OF 2
AD
A081856

END
DATE
FILMED
4-80
DTIC

GROW22

Same as GROW15, but with the number of periods T = 22.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1-21	20	43	356	41%	20	20	20	5%	3
22	20	43	356	41%					3
			7832	41%			420	5%	66

GRAND TOTALS

ROWS	440	(ALL EQUALITIES)
COLS	946	(748 BOUNDED)
ELEMS	8252	
DENS	2.0%	

COEFFICIENTS	NO SCALING
MAX ELEM	1.0
MIN ELEM	6.0×10^{-6}
LARGEST COL RATIO	1.3×10^{5}

SCSD8

A multi-stage structural design problem, documented in [29]. This is the only staircase test problem in which the stages do not represent periods of time.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1-38	10	70	130	19%	10	50	90	18%	70
39	17	90	224	15%					90
			5164	18%			3420	18%	2750

GRAND TOTALS

ROWS	397	(ALL EQUALITIES)
COLS	2750	
ELEMS	8584	
DENS	0.8%	

COEFFICIENTS	NO SCALING
MAX ELEM	1.0
MIN ELEM	2.4×10^{-1}
LARGEST COL RATIO	4.0

SCAGR25

Test problem reciefed from James K. Ho, Brookhaven National Laboratory, Upton, N.Y.; source not documented.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1	18	20	45	13%	8	7	17	30%	19
2-24	19	20	46	12%	8	7	17	30%	19
25	16	20	43	13%					19
			1146	12%			408	30%	475

GRAND TOTALS

ROWS	471	(300 EQUALITIES, 171 INEQUALITIES)
COLS	500	
ELEMS	1554	
DENS	0.7%	

COEFFICIENTS	NO SCALING
MAX ELEM	9.3
MIN ELEM	2.0×10^{-1}
LARGEST COL RATIO	1.9×10^{1}

SCRS8

Derived from a model of the United States' options for a transition from oil and gas to synthetic fuels; documented in [30,36].

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1	28	37	65	6%	25	22	29	5%	18
2	28	38	69	6%	25	22	29	5%	19
3-5	31	76	181	8%	25	22	29	5%	55
6-8	32	79	192	8%	25	22	29	5%	58
9	31	79	189	8%	25	22	29	5%	58
10-12	31	80	190	8%	25	22	29	5%	59
13-15	30	80	186	8%	25	22	29	5%	59
16	31	70	177	8%					59
			2747	8%			435	5%	847

GRAND TOTALS

ROWS	490	(384 EQUALITIES, 106 INEQUALITIES)
COLS	1169	
ELEMS	3182	
DENS	0.6%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	3.0×10^{2}	4.0
MIN ELEM	1.0×10^{-3}	2.5×10^{-1}
LARGEST COL RATIO	4.5×10^{3}	1.6×10^{1}

SCFXM2

Test problem received from James K. Ho, Brookhaven National Laboratory, Upton, N.Y.; source not documented.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1	92	114	679	6%	9	57	61	12%	13
2	82	99	434	5%	9	35	35	11%	4
3	66	126	300	4%	5	33	33	20%	1
4	90	118	1047	10%	5	5	5	20%	5
5	92	114	679	6%	9	51	61	12%	13
6	82	99	434	5%	9	35	35	11%	4
7	66	126	300	4%	5	33	33	20%	1
8	90	118	1047	10%					5
			4920	7%			263	13%	46

GRAND TOTALS

ROWS	660	(374 EQUALITIES, 286 INEQUALITIES)
COLS	914	
ELEMS	5183	
DENS	0.9%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	1.3×10^{2}	1.1×10^{1}
MIN ELEM	5.0×10^{-4}	8.7×10^{-2}
LARGEST COL RATIO	1.3×10^{5}	1.3×10^{2}

SCTAP2

A dynamic traffic assignment problem, documented in [31]. The LP has 11 objective rows; the objective named OBJZZZZZ was used in all tests. Statistics below omit the other ten objectives.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1-9	109	188	423	2%	62	138	276	3%	141
10	109	188	423	2%					141
			4230	2%			2484	3%	1410

GRAND TOTALS

ROWS	1090 (470 EQUALITIES, 620 INEQUALITIES)
COLS	1880
ELEMS	6714
DENS	0.3%

COEFFICIENTS	NO SCALING
MAX ELEM	8.0×10^1
MIN ELEM	1.0
LARGEST COL RATIO	8.0×10^1

PILOT

Derived from a welfare equilibrium model of the United States' energy supply, energy demand, and economic growth: seeks maximum aggregate consumer welfare subject to competitive market equilibrium. The LP was supplied by the PILOT modeling project, Systems Optimization Laboratory, Department of Operations Research, Stanford University; it is documented in [39].

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				SUB-STAIR BLOCKS		OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS	DENS	ELEMS
1	84	343	686	2%	31	74	105	5%	18	0%	10
2	90	345	1079	3%	34	76	111	4%	8	0%	10
3	90	343	1073	3%	34	74	109	4%	5	0%	10
4	90	343	1073	3%	34	74	109	4%	5	0%	10
5	90	343	1073	3%	34	74	109	4%	5	0%	10
6	90	343	1073	3%	34	74	109	4%	3	0%	10
7	90	343	1073	3%	32	74	107	5%	1	0%	10
8	87	341	1060	4%	4	19	19	25%			10
9	11	45	113	23%							12
			8303	3%			778	4%	45	0%	92

GRAND TOTALS

ROWS	722	(583 EQUALITIES, 139 INEQUALITIES)
COLS	2789	(80 FREE, 296 BOUNDED, 79 FIXED)
ELEMS	9126	
DENS	0.5%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	4.8×10^{4}	2.0×10^{1}
MIN ELEM	1.4×10^{-4}	4.9×10^{-2}
LARGEST COL RATIO	7.0×10^{6}	4.2×10^{2}

BP1

Developed by British Petroleum, London; supplied via the Systems Optimization Laboratory, Dept. of Operations Research, Stanford University.

This LP is approximately dual-angular, with six main diagonal blocks and about 400 coupling variables. For the experiments described in this paper it was treated as a 6-period, 6th-order staircase problem.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				SUB-STAIR BLOCKS		OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS	DENS	ELEMS
1	111	227	1400	6%	3	60	3	2%	163	0%	138
2	151	353	2175	4%	62	108	112	2%	142	0%	149
3	113	321	964	3%	92	232	346	2%	494	1%	270
4	170	295	2178	4%	51	14	11	2%	4	0%	74
5	134	198	1315	5%	111	2	2	1%			40
6	142	177	1091	4%							56
			9123	4%			474	2%	803	0%	727

GRAND TOTALS

ROWS	821	(516 EQUALITIES, 305 INEQUALITIES)
COLS	1571	
ELEMS	10400	
DENS	0.8%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	2.4×10^{2}	1.3×10^{1}
MIN ELEM	2.0×10^{-4}	7.6×10^{-2}
LARGEST COL RATIO	1.7×10^{5}	1.7×10^{2}

APPENDIX B: DETAILS OF COMPUTATIONAL TESTS

Computing environment

All computational experiments were performed on the Triplex system [57] at the Stanford Linear Accelerator Center, Stanford University. The Triplex comprises three computers linked together: one IBM 360/91, and two IBM 370/168s. Runs were submitted as batch jobs in a virtual-machine environment, under the control of IBM systems OS/VS2, OS/MVT and ASP.

Each group of test runs employed a specially-modified version of the linear-programming routines from the MINOS system [38,47]. MINOS is written in standard FORTRAN. For the timed runs MINOS was compiled with the IBM FORTRAN IV (H extended, enhanced) compiler, version 1.1.0, at optimization level 3 [32].

Timings

All running-time statistics are based on "CPU second" totals for individual job steps as reported by the operating system. To promote consistency all timed jobs were run on the Triplex computer designated "system A," and jobs whose timings would be compared were run at about the same time. Informal experiments indicated roughly a 1% variation in timings due to varying system loads.

More detailed timings employed PROGLOOK [33], which takes frequent samples of a running program to estimate the proportion of time spent in each subroutine. To determine the actual time in seconds for each subroutine, a job was run twice--once without PROGLOOK to measure total CPU seconds, and once with PROGLOOK to estimate each subroutine's proportion of the total. PROGLOOK estimates were based on at least 2300 samples per job.

Linear-programming environment

MINOS was set up for test runs according to the defaults indicated in [38], with the exception of the items listed below.

Scaling. Problems noted as "scaled" in Appendix A were subjected to the following geometric-mean scaling (where A denotes the matrix of constraint coefficients, not including the objective or right-hand side):

1: Compute $\rho_0 = \max|A_{i_1 j}/A_{i_2 j}|$, $A_{i_2 j} \neq 0$.

2: Divide each row i of A, and its corresponding right-hand side value, by $[(\min_j |A_{ij}|)(\max_j |A_{ij}|)]^{1/2}$, taking the minimum over all $A_{ij} \neq 0$.

3: Divide each column j of A, and its corresponding coefficient in the objective, by $[(\min_i |A_{ij}|)(\max_i |A_{ij}|)]^{1/2}$, taking the minimum over all $A_{ij} \neq 0$.

4: Compute $\rho = \max|A_{i_1 j}/A_{i_2 j}|$, $A_{i_2 j} \neq 0$.

The procedure is repeated as many times as possible until, at step 4, ρ is at least 90% of ρ_0. (In other words, scaling continues as long as it reduces ρ, the greatest ratio of two elements in the same column, by more than 10%.)

Starting basis. Runs that are said below to be from an "all-slack start" employed crash option 0 of MINOS: the initial basis was composed entirely of unit vectors, and all nonbasic variables were placed at their lower (or only) bounds. Runs from an "advanced start" used an initial basis that had been reached and saved in a previous MINOS run.

Pricing. Except for SCTAP2, e partial-pricing scheme of MINOS was employed--with one important change: the arbitrary partitioning of the columns normally defined by MINOS for partial pricing was replaced by the natural staircase partition. Thus the periods of the staircase were priced one at a time in a cyclic fashion.

Pricing for SCTAP2 was similar except that the incoming column was chosen from the lastest possible period. (This choice was known to produce a relatively small number of iterations from an all-slack start.)

Refactorization frequency. MINOS was instructed to refactorize the basis (by performing a fresh Gaussian elimination) every 50 iterations, except for BP1 (every 75) and PILOT (every 90).

Elimination tolerance. The "LU ROW TOL" for MINOS--which is essentially the value δ defined in Section 1--was set to 10^{-4}. Other tolerances were left at their default values.

Test group 1

The linear programs in test group 1 were used to investigate fill-in and stability of structural elimination techniques, as described in Sections 3 and 5.

The special version of MINOS for test group 1 retained MINOS' original routines for standard bump-and-spike elimination. New routines were added to implement a version of staircase bump-and-spike elimination, and to compile statistics. Additionally, control routines were modified so that Gaussian elimination was carried out twice at each refactorization, once each with the standard and staircase techniques.

Briefly, the new routines for this version were as follows:

SP3--an adaptation of the P3 heuristic to find spikes non-square or rank-deficient blocks, as proposed in Section 5. This routine is a modification of the original MINOS subroutine P3.

SP4--main routine for the staircase bump-and-spike pivot-selection method of Section 5; calls SP3 once for each diagonal block of the staircase.

MAP--a basis analyzer: permutes the basis to reduced standard form (Section 2) and reports statistics on the shape and density of staircase blocks.

In addition three routines were modified to efficiently handle either a standard or staircase bump-and-spike pattern: INVERT, the main routine for refactorization; FACTOR, an implementation of Gaussian elimination (using the preselected pivots); and DSPSPK, a utility routine that prints a display of the spike structure.

For most runs the staircase technique was applied last, and the resulting factors were used in the following simplex iterations. However, SCSD8 was also run with the standard technique last, as the factorization from the standard technique yielded a quite different iteration path.

All LPs except PILOT and BP1 were run from an all-slack start to optimality. PILOT and BP1 were run 1000 and 750 iterations, respectively, from advanced starts.

Test group 2

The linear programs in test group 2 were used to investigate fill-in and stability of elimination techniques that perform a local minimization, as described in Sections 3 and 6.

The special version of MINOS for test group 2 was an augmentation of the version for test group 1. Routines were added to implement three local-minimization strategies: standard, full-staircase, and reduced-staircase. Control routines were altered so that Gaussian elimination was carried out five times at each refactorization, once with each of the above strategies and once each with the standard and staircase bump-and-spike techniques.

The routines added for local-minimization strategies were briefly as follows:

MAP--a basis analyzer: permutes the basis to reduced standard form (Section 2), reports statistics on the shape and density of staircase blocks, and constructs a "bit map" that is used to keep track of non-zero elements during the elimination.

FACMER--an implementation of sparse Gaussian elimination by local-minimization techniques, employing $(r_i^{(k)} - 1)(c_j^{(k)} - 1)$ as the merit function. This routine can be instructed to use the standard, full-staircase or reduced-staircase strategy. It searches rows and columns for the minimizing element in increasing order of $r_i^{(k)}$ and $c_j^{(k)}$; columns are searched before rows of equal count. Stability of the selected pivot is determined by a column-ratio test only, as explained in Section 6.

INVMER--main routine for the local-minimization techniques, based on the MINOS subroutine INVERT. INVMER calls FACMER three times--once for each strategy--and handles storage allocation and solution checks.

DRIVER, the main LP routine of MINOS, was modified to call both INVERT and INVMER and to collect and print the results.

FACMER was intended as a quick and simple implementation of local-minimization methods using MINOS' existing storate schemes; hence it is fairly slow and does not make the most efficient use of storage. A sophisticated implementation of FACMER would require much more extensive modifications to MINOS.

For most runs the staircase bump-and-spike technique was applied last, and the resulting factors were used in the following simplex iterations. SCSD8 was also run with the standard bump-and-spike last as for test group 1.

GROW15 was run from an all-slack start to optimality. The other LPs were run for 240 CPU seconds from an all-slack start, and also from one or more advanced starts.

Test group 3

The linear programs of test group 3 were used to make timings of structural techniques for sparse elimination, as reported in Section 5.

The special version of MINOS for test group 3 employed the same SP3, SP4, FACTOR and INVERT routines as test group 1, but only one elimination--either standard or staircase--was carried out at each refactorization. Two different runs were made for each LP, one using the standard technique only and one using the staircase technique only. Starting bases and run lengths were as for test group 1.

Further details of this test group are presented in [22], which reports much more extensive results from the same timing runs.

APPENDIX C: ADDITIONAL STATISTICS FROM TEST GROUP 1

Sparsity

All figures below are averages over sets of bases, as explained in Section 3; values under # indicate how many bases are in each set. ROWS is the number of basis rows (excluding the objective), and STR COLS is the number of "structural" columns in the basis--remaining columns are unit vectors corresponding to slack variables.

Figures under ELEMENTS are numbers of nonzeroes in the basis B in reduced standard staircase form. Separate totals are given for the diagonal blocks B_{kk} (ON), the off-diagonal blocks $\hat{B}_{k+1,k}$ (OFF), the sub-staircase blocks $B_{k+2,k}$ through B_{tk} (SUB), and the entire basis (TOT). Under % are precentage densities for the diagonal and off-diagonal blocks.

			STR	ELEMENTS					
	#	ROWS	COLS	ON	%	OFF	%	SUB	TOT
GROW15	3	300	207	2146	36%	83	12%	---	2229
GROW22	5	440	398	4178	47%	150	12%	---	4328
SCSD8-	21	397	383	665	16%	436	19%	---	1101
SCSD8+	11	397	391	684	16%	451	19%	---	1135
SCSD8++	12	397	393	688	17%	443	19%	---	1131
SCAGR25-	12	471	362	1023	12%	354	38%	---	1377
SCAGR25+	9	471	338	1005	11%	343	40%	---	1348
SCRS8-	12	490	369	915	6%	326	8%	---	1241
SCRS8+	4	490	350	911	6%	309	9%	---	1220
SCFXM2-	11	660	444	2282	4%	126	16%	---	2408
SCFXM2+	5	660	464	2366	4%	139	15%	---	2505
PILOT-	3	722	690	2661	4%	294	5%	25	2980
PILOT+	9	722	688	2660	4%	291	5%	26	2977
BP1-	4	821	655	3971	4%	185	2%	274	4430
BP1+	6	821	656	3987	4%	190	2%	235	4412

Structure

The first values below are the numbers of periods (PER) and lower square sub-staircases (LOW SSS) in the staircase bases.

Remaining figures refer to the block-triangular reductions of the bases: number of bumps (BUMPS), number of columns in the largest bump (BIG) and in the three largest bumps (3 BIG), and number of spikes in all bumps (SPKS).

		LOW		COLS IN BUMPS		
	PER	SSS	BUMPS	BIG	3 BIG	SPKS
GROW15	15	5	4	151	172	92
GROW22	22	2	7	171	304	183
SCSD8-	39	4	16	83	129	28
SCSD8+	39	2	15	133	192	34
SCSD8++	39	0	14	180	224	36
SCAGR25-	25	2	7	27	45	7
SCAGR25+	25	2	4	78	106	9
SCRS8-	16	1	13	33	67	19
SCRS8+	16	1	6	117	139	16
SCFXM2-	8	1	21	36	82	32
SCFXM2+	8	1	18	63	119	40
PILOT-	9	-	9	455	514	94
PILOT+	9	-	5	538	547	92
BP1-	6	-	10	316	414	118
BP1+	6	-	9	417	429	122

APPENDIX D: ADDITIONAL STATISTICS FROM TEST GROUP 2

Sparsity

All figures below are averages over sets of bases, as explained in Section 3; values under # indicate how many bases are in each set. ROWS is the number of basis rows (excluding the objective), and STR COLS is the number of "structural" columns in the basis--remaining columns are unit vectors corresponding to slack variables.

Figures under ELEMENTS are numbers of nonzeroes in the basis B in reduced standard staircase form. Separate totals are given for the diagonal blocks B (ON), the off-diagonal blocks $\hat{B}_{k+1,k}$ (OFF), and the entire basis (TOT). Under % are percentage densities for the diagonal and off-diagonal blocks.

			STR	ELEMENTS				
	#	ROWS	COLS	ON	%	OFF	%	TOT
GROW15	3	300	207	2146	36%	83	12%	2229
SCSD8-	4	397	342	626	15%	401	20%	1027
SCSD8+	4	397	396	685	17%	441	19%	1126
SCAGR25-	3	471	304	1013	11%	299	54%	1312
SCAGR25+	2	471	324	1028	12%	320	48%	1348
SCRS8-	6	490	364	917	6%	319	8%	1236
SCRS8+	3	490	347	909	6%	307	9%	1216
SCFXM2-	4	660	395	2086	4%	108	17%	2194
SCFXM2+	2	660	458	2331	4%	138	15%	2469

Structure

The first values below are the numbers of periods (PER) and lower square sub-staircases (LOW SSS) in the staircase bases.

Remaining figures refer to the block-triangular reductions of the bases: number of bumps (BUMPS), number of columns in the largest bump (BIG) and in the three largest bumps (3 BIG), and number of spikes in all bumps (SPKS). (Entries marked * showed too much variation to be meaningfully averaged.)

		LOW		COLS IN BUMPS		
	PER	SSS	BUMPS	BIG	3 BIG	SPKS
GROW15	15	5	4	151	172	92
SCSD8-	39	4	15	88	118	28
SCSD8+	39	3	15	159	205	35
SCAGR25-	25	2	*	18	29	10
SCAGR25+	25	0	*	124	155	14
SCRS8-	16	2	12	41	76	18
SCRS8+	16	0	6	124	144	15
SCFXM2-	8	2	21	39	88	32
SCFXM2+	8	1	19	54	123	41

REFERENCES

[1] Alway, G. G. and D. W. Martin, "An Algorithm for Reducing the Bandwidth of a Matrix of Symmetrical Configuration." Computer Journal 8 (1965), 264-272.

[2] Bartels, Richard H., "A Stabilization of the Simplex Method." Numerische Mathematik 16 (1971), 414-434.

[3] _____ and Gene H. Golub, "The Simplex Method of Linear Programming Using LU Decomposition." Communications of the ACM 12 (1969), 266-268.

[4] Beale, E. M. L., "Sparseness in Linear Programming." Large Sparse Sets of Linear Equations, J. K. Reid, ed. (New York: Academic Press, 1971), 1-15.

[5] Bertelè, Umberto and Francesco Brioschi, "A Note on a Paper by Spillers and Hickerson." Quarterly of Applied Mathematics 24 (1971), 311-313.

[6] Brayton, Robert K., Fred G. Gustavson, and Ralph A. Willoughby, "Some Results on Sparse Matrices." Mathamtics of Computation 24 (1970), 937-954.

[7] Chen, Y. T. and R. P. Tewarson, "On the Optimal Choice of Pivots for the Gaussian Elimination." Computing 9 (1972), 245-250.

[8] Curtis, A. R. and J. K. Reid, "The Solution of Large Sparse Unsymmetric Systems of Linear Equations." Information Processing 71 (North-Holland Publishing Co., 1972), 1240-1245.

[9] Dantzig, George B. et al., "Sparse Matrix Techniques in Two Mathematical Programming Codes." Proceedings of Sparse Matrix Symposium, R. A. Willoughby, ed. (IBM Watson Research Center, 1969), 85-99.

[10] Douglas, Allan, "Examples Concerning Efficient Strategies for Gaussian Elimination." Computing 8 (1971), 382-394.

[11] Duff, Iain S., "On the Number of Nonzeroes Added when Gaussian Elimination is Performed on Sparse Random Matrices." Mathematics of Computation 28 (1974), 219-230.

[12] _____, "On Algorithms for Obtaining a Maximum Transversal." Report CSS 49, Computer Science and Systems Division, A.E.R.E. Harwell, England (1978).

[13] _____, "Practical Comparisons of Codes for the Solution of Sparse Linear Systems." Sparse Matrix Proceedings-1978, Iain S. Duff and G. W. Stewart, eds. (Society for Industrial and Applied Mathematics, 1979).

[14] _____ and J. K. Reid, "A Comparison of Sparsity Orderings for Obtaining a Pivotal Sequence in Gaussian Elimination." Journal of the Institute of Mathematics and Its Applications 14 (1974), 281-291.

[15] _____ and J. K. Reid, "Performance Evaluation of Codes for Sparse Matrix Problems." Report CSS 66, Computer Science and Systems Division, A.E.R.E. Harwell, England (1978).

[16] _____ and J. K. Reid, "An Implementation of Tarjan's Algorithm for the Block-Triangularization of a Matrix." ACM Transactions on Mathematical Software 4 (1978), 137-147.

[17] _____ and J. K. Reid, "Some Design Features of a Sparse Matrix Code." ACM Transactions on Mathematical Software 5 (1979), 18-35.

[18] Dulmage, A. L. and N. S. Mendelsohn, "Coverings of Bipartite Graphs." Canadian Journal of Mathematics 10 (1958), 517-534.

[19] _____, "A Structure Theory of Bipartite Graphs of Finite Exterior Dimension." Transactions of the Royal Society of Canada (series 3) 53, sect. 3 (1959), 1-13.

[20] _____, "On the Inversion of Sparse Matrices." Mathematics of Computation 16 (1962), 494-496.

[21] _____, "Two Algorithms for Bipartite Graphs." SIAM Journal 11 (1963), 183-194.

[22] Fourer, Robert, "Solving Staircase Linear Programs by the Simplex Method, 1: Inversion." Technical Report Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1979).

[23] Gay, David M., "On Combining the Schemes of Reid and Saunders for Sparse LP Bases." Sparse Matrix Proceedings-1978, Iain S. Duff and G. W. Stewart, eds. (Society for Industrial and Applied Mathematics, 1979).

[24] Gear, C. W. et al., "Numerical Computation: Its Nature and Research Directions." SIGNUM Newsletter (Association for Computing Machinery, 1979).

[25] Glassey, C. Roger and Peter Benenson, "A Quadratic Programming Analysis of Energy in the United States Economy." Report ES-116, Electric Power Research Institute, Palo Alto, CA (1975).

[26] Gustavson, Fred, "Finding the Block Lower Triangular Form of a Sparse Matrix." Sparse Matrix Computations, James R. Bunch and Donald J. Rose, eds. (Academic Press, 1976), 275-289.

[27] Hellerman, Eli and Dennis Rarick, "Reinversion With the Preassigned Pivot Procedure." Mathematical Programming 1 (1971), 195-216.

[28] _____, "The Partitioned Preassigned Pivot Procedure (P^4)." Sparse Matrices and Their Applications, Donald J. Rose and Ralph A. Willoughby, eds. (New York: Plenum Press, 1972), 67-76.

[29] Ho, James K., "Optimal Design of Multi-Stage Structures: A Nested Decomposition Approach." Computers and Structures 5 (1975), 249-255.

[30] _____, "Nested Decomposition of a Dynamic Energy Model." Management Science 23 (1977), 1022-1026.

[31] _____, A Successive Linear Optimization Approach to the Dynamic Traffic Assignment Problem." Report BNL-24713, Brookhaven National Laboratory, Upton, N.Y. (1978).

[32] IBM OS FORTRAN IV (H Extended) Compiler Programmer's Guide. No. SC28-6852, International Business Machines Corp. (1974).

[33] Johnson, R. and T. Johnston, "PROGLOOK User's Guide." User Note 33, SLAC Computing Services, Stanford Linear Accelerator Center (1976).

[34] Kalan, James E., "Aspects of Large-Scale In-Care Linear Programming." Proceedings of the ACM (1971), 304-313.

[35] Keller, Herbert B., "Accurate Difference Methods for Nonlinear Two-Point Boundary Value Problems." SIAM Journal on Numerical Analysis 11 (1974), 305-320.

[36] Manne, Alan S., "U. S. Options for a Transition from Oil and Gas to Synthetic Fuels." Disscussion Paper 26D, Public Policy Program, Kennedy School of Government, Harvard University (1975).

[37] Markowtiz, Harry M., "The Elimination Form of the Inverse and Its Application to Linear Programming." Management Science 3 (1957), 255-269.

[38] Murtagh, Bruce A. and Michael A. Saunders, "MINOS: A Large-Scale Nonlinear Programming System (For Problems with Linear Constraints): User's Guide." Technical Report SOL 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University (1977).

[39] Parikh, S.C., "A Welfare Equilibrium Model (WEM) of Energy Supply, Energy Demand, and Economic Growth," Technical Report SOL 79-3, Systems Optimization Laborartory, Dept. of Operations Research, Stanford University (1979).

[40] Parter, S., "The Use of Linear Graphs in Gauss Elimination." SIAM Review 3 (1961), 119-130.

[41] Propoi, A. and V. Krivonozhko, "The Simplex Method for Dynamic Linear Programs." Report RR-78-74, International Institute for Applied Systems Analysis, Laxenburg, Austria (1978).

[42] Reid, J. K., "A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases." Report CSS 20, Computer Science and Systems Division, A.E.R.E. Harwell, England (1975).

[43] Rose, Donald J. and Robert Endre Tarjan, "Algorithmic Aspects of Vertex Elimination on Directed Graphs." SIAM Journal on Applied Mathematics 34 (1978), 176-197.

[44] Saigal, Romesh, "Block-Triangularization of Multi-Stage Linear Programs." Report ORC 66-9, Operations Research Center, University of California, Berkeley (1966).

[45] Saunders, Michael A., "The Complexity of LU Updating in the Simplex Method." The Complexity of Computational Problem Solving, R. S. Anderssen and R. P. Brent, eds. (Queensland University Press, 1975), 214-230.

[46] _____, "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating." Sparse Matrix Computations, James R. Bunch and Donald J. Rose, eds. (Academic Press, 1976), 213-226.

[47] _____, "MINOS System Manual." Technical Report SOL 77-31, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1977).

[48] Steward, Donald V., "Partitioning and Tearing Systems of Equations." SIAM Journal on Numerical Analysis 2 (1965), 345-365.

[49] Tarjan, Robert, "Depth-First Search and Linear Graph Algorithms." SIAM Journal on Computing 1 (1972), 146-160.

[50] Tewarson, R. P., "Solution of a System of Simultaneous Linear Equations with a Sparse Coefficient Matrix by Elimination Methods." BIT 7 (1967), 226-239.

[51] _____, "On the Product Form of Inverses of Sparse Matrices and Graph Theory." SIAM Review 9 (1967), 91-99.

[52] _____, "On the Gaussian Elimination Method for Inverting Sparse Matrices." Computing 9 (1972), 1-7.

[53] _____ and K. Y. Cheng, "A Desirable Form for Sparse Matrices when Computing their Inverse in Factored Forms." Computing 11 (1973), 31-38.

[54] Tomlin, J. A., "Pivoting for Size and Sparsity in Linear Programming Inversion Routines." Journal of the Institute of Mathematics and Its Applications 10 (1972), 289-295.

[55] Varah, J. M., "On the Solution of Block-Tridiagonal Systems Arising from Certain Finite-Difference Equations." Mathematics of Computation 26 (1972), 859-868.

[56] _____, "Alternate Row and Column Elimination for Solving Certain Linear Systems." SIAM Journal on Numerical Analysis 13 (1976), 71-75.

[57] Vinson, Ilse, "Triplex User's Guide." User Note 99, SLAC Computing Services, Stanford Linear Accelerator Center (1968).

[58] Weil, R. L. and P. C. Kettler, "Rearranging Matrices to Block-Angular Form for Decomposition (and Other) Algorithms." Management Science 18 (1971), 98-108.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SOL 79-17	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Sparse Gaussian Elimination of Staircase Linear Systems		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER SOL 79-17
7. AUTHOR(s) Robert Fourer		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0267
9. PERFORMING ORGANIZATION NAME AND ADDRESS Operations Research Department - SOL Stanford University Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-143
11. CONTROLLING OFFICE NAME AND ADDRESS Operations Research Program -- ONR Department of the Navy 800 N. Quincy St., Arlington, VA 22217		12. REPORT DATE November 1979
		13. NUMBER OF PAGES 118
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

LINEAR ALGEBRA	LARGE-SCALE SYSTEMS
NUMERICAL ANALYSIS	SPARSE SYSTEMS
ELIMINATION	
STAIRCASE SYSTEMS	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SOL 79-17, Robert Fourer
SPARSE GAUSSIAN ELIMINATION OF STAIRCASE LINEAR SYSTEMS

A square system of linear equations is said to be sparse if it can be solved most efficiently through a knowledge of its arrangement of zero and nonzero coefficients. Sparse systems are commonly solved by the techniques of sparse Gaussian elimination. An important class of sparse systems are those that have a "staircase" structure: their variables fall into a natural sequence of disjoint groups, and each equation relates only variables within one group or within two adjacent groups.

This paper proposes special methods of sparse Gaussian elimination for staircase-structured systems. These methods are particularly applicable to linear programming problems whose constraints have a staircase structure; they may also find application in solving staircase linear systems that arise in nonlinear optimization and optimal control.

The initial sections of this paper present a self-contained review of sparse elimination, and derive pertinent properties of staircase systems. Subsequent sections pursue two approaches to staircase elimination, and report initial computational experience in detail.